MASTER EN SCIENCES INFORMATIQUES

Proposition de modélisation des services bioinformatiques dans le cadre d 'une architecture fédérée

Denayer, Marie-Laetitia

Award date:
2004

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année Académique 2003 - 2004

Proposition de modélisation des services bioinformatiques dans le cadre d'une architecture fédérée

Denayer Marie-Laetitia

Mémoire présenté en vue de l'obtention du grade de Maître et Licencié en Informatique.

Résumé

Depuis les années 80, le biologiste dispose d'une nouvelle manière de réaliser des expériences : la bioinformatique. De plus, la multiplication des sites Internet proposant des services bioinformatiques a fait de cet espace virtuel un immense laboratoire.

Malheureusement, les collaborations entre ces différents sites se limitent à un échange de données. Très peu de tentatives existent pour uniformiser l'utilisation et l'aspect de sites. Les sites sont donc hétérogènes. Or, le biologiste est amené à faire collaborer ces sites. Il doit donc gérer cette hétérogénéité, ce qui est un frein dans son utilisation de ce laboratoire. De plus, l'utilisation de ces sites est tellement complexe que le biologiste éprouve une certaine appréhension envers la bioinformatique.

Pourtant des solutions informatiques existent pour gérer l'hétérogénéité, tels les systèmes distribués. Quelques projets émergent d'ailleurs pour réaliser des systèmes distribués dans le cadre de la bioinformatique tel BIGRE qui est réalisé en partenariat entre les FUNDP et l'ULB.

Ce mémoire s'inscrit dans le cadre de ce projet et a pour but la modélisation des différents éléments constitutifs de la bioinformatique en vue de proposer au biologiste une interface utilisateur homogène.

Abstract

Since the eighties, the biologist has a new manner of carrying out experiments: bioinformatics. Moreover, Internet gave the biologist the opportunity of having a direct access to this large laboratory distributed on various sites.

Unfortunately, collaborations between these various sites are restricted to data exchange. Few attempts exist to standardize the data format, the appearance of the site, etc. The sites are thus heterogeneous. However, the biologist must sometimes make the sites collaborate. He must thus manage this heterogeneity, that curbs its use of bioinformatics.

However IT solutions to manage this heterogeneity exists like the distributed systems. Some projects emerge in order to develop distributed systems in the bioinformatics framework. One of them that is called BIGRE is carried out in partnership between the University of Brussels and the University of Namur.

This master thesis is within the scope of this project and its aim is the modeling of various components of bioinformatics in order to offer a homogeneous user interface to the biologist.

Je tiens à remercier avant tout les personnes qui m'ont aidée à réaliser ce mémoire.

Tout d'abord, mon promoteur, M. Englebert, qui a sans cesse su m'encourager et me donner des conseils avisés.

Je tiens également à remercier tous les membres de l'IBMM et de l'EBI pour m'avoir permis de m'immerger dans l'univers de la bioinformatique. Je pense plus particulièrement à Valérie Ledent, Richard Kamuzinzi et Viorica Ghita qui ont pris le temps de répondre à mes nombreuses questions.
Ensuite, je remercie mon maître de stage Marc Colet pour m'avoir trouvé un stage de six semaines en Angleterre à l'EBI.
Je remercie également l'équipe de BIGRE pour les confrontations d'idée, plus particulièrement Pierre Buyle et Quentin Dallons.

Je pense également à toutes les personnes qui ont pris le temps de lire ce mémoire pour me donner leur avis : Liliane, Quentin, Christophe, André. Je tiens également à remercier Geoffrey et Jessica pour leurs commentaires tout au long de ma rédaction, ainsi que Magali pour ses corrections du résumé en anglais.

Je remercie enfin tous mes amis et ma famille qui m'ont supporté pendant la rédaction de ce mémoire.

Table des matières

Table des figures

Liste des tableaux

Introduction

La bioinformatique est une branche de la biologie moléculaire qui utilise l'informatique pour réaliser des expériences : d'abord, au moyen de banques de données stockant la majorité des informations découvertes à ce jour, ensuite, au moyen de programmes permettant la recherche d'informations, la comparaison de séquences ou encore la prédiction de propriétés biologiques,...

La bioinformatique s'est appuyée sur le développement d'Internet pour permettre à tous les biologistes d'accéder aux différentes données et aux différents programmes disponibles à travers le monde.

Néanmoins, Internet ne permet pas de réaliser aisément certaines opérations pourtant importantes comme la recherche d'un programme ou la réalisation d'un enchaînement de programmes (parfois sur des sites distants) car aucun outil spécifique n'aide le biologiste dans ces tâches.

Des solutions logicielles issues des technologies de l'information et de la communication existent pour faciliter le déploiement et la collaboration des systèmes d'information (par exemple : OMG CORBA, Services WEB, etc.). Utiliser ces solutions dans le cadre de la bioinformatique permettrait aux utilisateurs d'accéder via une interface unique à tous les services qui permettrait d'offrir à tous les biologistes des outils réalisant la recherche et l'enchaînement de ceux-ci.

De plus , les ressources de bioinformatique sont hétérogènes. D'une part, on trouve des ensembles de programmes amis bien intégrés où l'utilisateur peut aisément faire interagir ces programmes. D'autre part, on trouve des programmes solitaires, souvent réalisés pour un groupe de laboratoires.

L'utilisateur peut accéder à ces différents programmes ou à des services (sous-ensemble de fonctionnalités d'un programme) au moyen de page web. Néanmoins, ces interfaces (site web, interface graphique, etc.) sont également hétérogènes : elles ne proposent pas la même information pour un même service et la même information n'est pas présentée de la même façon.

L'utilisateur doit aujourd'hui s'adapter à l'interface proposée et donc au service proposée. Le système distribué permettrai également de gérer cette hétérogénéité et proposerai à l'utilisateur une interface homogène.

De nombreux projets (aussi bien publics que privés) tentent de réaliser un tel système distribué. Parmi ceux-ci, le projet BIGRE est en cours de développement pour mettre à la disposition des différents utilisateurs (bioinformaticiens, biologistes, cliniciens, etc.) un

large système distribué et fédéré d'accès à des services bioinformatiques. Il est le fruit d'une collaboration inter-universitaire entre l'Institut d'informatique des FUNDP, l'Unité de Bioinformatique du département de Biologie Moléculaire de l'ULB (IBMM) et l'Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle de l'ULB (IRIDIA).

Dans le cadre de ce projet, ce mémoire propose une modélisation des différents éléments constitutifs de la bioinformatique : *donnée*, *service*, *interface graphique* et *enchaînement* afin de faciliter l'intégration de ces éléments dans le système distribué.

Pour cerner les habitudes des utilisateurs et des auteurs de programmes, un stage fut réalisé dans deux institutions : une première partie à l'Institut de Biologie et de Médecine Moléculaire de l'ULB (IBMM) auprès de professeurs et d'utilisateurs de bioinformatique, une seconde partie à l'European Bioinformatic Institute (EBI) près de Cambridge, auprès des dévellopeurs du package EMBOSS (qui regroupe une série de programmes de bioinformatique).

Nous avons divisé ce mémoire en deux parties.

La première partie présentera toutes les informations utiles aux lecteurs pour comprendre les enjeux de la bioinformatique ainsi que les difficultés rencontrées par les biologistes dans la réalisation de leurs expériences. Dans le premier chapitre, nous mettrons en évidence les différents éléments constituant la bioinformatique. Le second chapitre mettra l'accent sur les difficultés de l'utilisation des services actuels et sur les apports qu'offrirait un système distribué.

La deuxième partie couvrira la modélisation réalisée. Nous présenterons dans le troisième chapitre l'intérêt de cette modélisation ainsi que l'articulation des différents éléments de cette modélisation. Dans le chapitre suivant, nous aborderons la modélisation des données bioinformatiques. Nous aborderons ensuite dans le cinquième et sixième chapitre la modélisation des services et de l'IHM. Dans le septième chapitre, nous mettrons en évidence un façon de représenter les enchaînements de services (ou protocoles). Enfin dans le dernier chapitre, nous envisagerons quelques extensions possibles à notre modélisation.

Première partie

Mise en place du décor

Chapitre 1

Bioinformatique

Ce chapitre a été écrit sur la base de la compréhension de la bioinformatique que nous avons acquise lors de notre stage à l'IBMM, ainsi que suite aux lectures suivantes : [Claverie et Notredame, 2003, BEN, 2004b, BEN, 2004a]

Après une brève introduction relative à la biologie, nécessaire pour comprendre les enjeux de ce travail, nous étudierons la bioinformatique et présenterons une utilisation particulière.

1.1 Un peu de biologie

Tous les organismes sont constitués de cellules. Ainsi le corps humain est composé d'environ $6 * 10^{13}$ cellules de 320 types différents comme les cellules de la peau, des muscles, des neurones (cellule du cerveau), etc.

La cellule est constituée principalement d'eau (70%) et de protéines (environ 20%), les 10% restants sont des petits organes comme le noyau (sert au contrôle général de la cellule), la mitochondrie (permet la respiration de la cellule), etc. Nous allons plus particulièrement nous intéresser :

- aux **protéines** qui sont les blocs constructeurs et les molécules fonctionnelles de la cellule
- à l'**ADN** qui stocke l'information propre à nos différentes caractéristiques (couleur des yeux, etc.). Cette information est présente dans chaque cellule au niveau du noyau pour aider au contrôle de la cellule.

1.1.1 Les protéines

Toutes les protéines sont réalisées à partir des mêmes constituants : les acides aminés. Ce sont des molécules relativement complexes composées d'atomes de carbone, d'hydrogène, d'oxygène, d'azote et de souffre. Ainsi la composition d'une protéine ressemble à $C_{1200}H_{2400}O_{600}N_{300}S_{100}$.

Les biochimistes ont tenté de trouver une représentation des protéines, de préférence une formule qui pourrait également expliquer leurs propriétés biologiques. Une protéine est constituée d'un grand nombre d'acides aminés (entre 100 et 500) choisis parmi une sélection de 20 goûts (chaque type est représenté par une lettre). Or chaque type de protéine contient toujours le même nombre de chaque acide aminé. La formule d'une protéine peut donc être

exprimée par le nombre d'acides aminés qu'elle contient. Par exemple, la formule de l'insuline est : 30 glycine(G) + 40 alanine(A) + 5 tyrosine(T) + 14 glutamine(Q) et d'autres acides aminés.

Ensuite, les biologistes ont découvert que ces acides aminés étaient liés comme une chaîne. L'identité de la protéine ne dépend plus seulement du nombre d'acides aminés mais également de leur ordre. La formule de l'insuline fut découverte en 1951 et est représentée par cette chaîne : MALWMRPLLPLLALLA...ENYCN.

La suite d'acides aminés détermine l'identité d'une protéine mais la forme que la protéine adopte dans son environnement permet de connaître la fonction de cette protéine, ses propriétés biologiques (comme l'aptitude à digérer le sucre), etc.

Cette forme est encodée dans la chaîne. En effet, certains acides sont hydrophobes et donc ne veulent pas interagir avec la surface proche de l'eau, d'autres hydrophiles recherchent cette opportunité. D'autres éléments ont également un impact sur la forme comme la charge électrique, etc.

La forme d'une protéine (comme l'ubiquitin, voir figure 1.1) peut également être considérée comme un assemblage de structures[1]. La structure qualifie le type de forme adoptée par un segment d'acides aminés. Cette forme provient des liaisons chimiques entre les acides aminés et possède certaines propriétés telle la stabilité. Dans le schéma, on reconnaît deux structures : une hélice α (en rouge), un feuillet β (en bleu). Ces deux structures sont les plus stables[2] connues aujourd'hui.

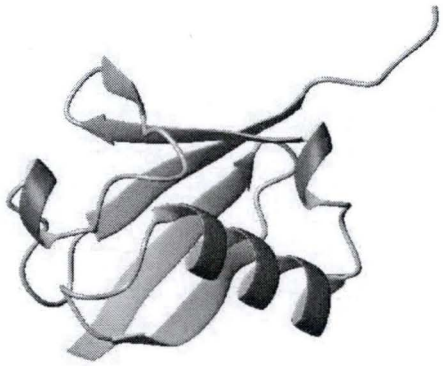


FIG. 1.1 – *Structure 3D d'une protéine*
source : `http://darwin.chem.nottingham.ac.uk/group/protein.html`

La protéine est construite sur base des informations stockées dans l'ADN, nous verrons comment dans la section suivante.

1. Pour être plus précis, on appelle ce type de structure *structure secondaire*. La *structure primaire* est la chaîne d'acides aminés. La *structure tertiaire* est la forme 3D de la protéine. La *structure quaternaire* est la forme prise par l'assemblage de plusieurs protéines.
2. Ces deux structures changent rarement de forme.

1.1.2 L'ADN

Nos caractéristiques comme notre couleur de cheveux, etc. constituent notre information génétique, celle-ci est repartie sur 46 chromosomes. Ces chromosomes sont notre matériel génétique aussi appelé génome. Un chromosome (voir figure 1.2) est composé d'ADN compacté, il provient soit de notre père, soit de notre mère.

L'ADN est composé de deux chaînes (ou brins) de nucléotides formant une double hélice (voir figure 1.2). Ces nucléotides peuvent prendre quatre valeurs : A, T, G, C. La portion d'ADN codant pour une protéine est appelé gène.

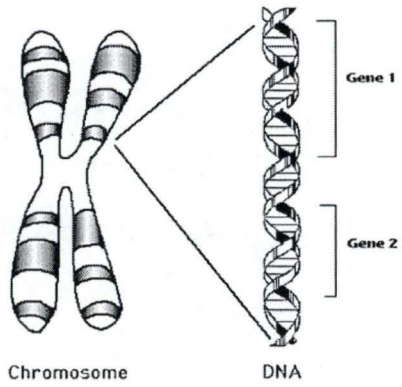


FIG. 1.2 – *Forme en double hélice de l'ADN*
source : `http://www.stedwards.edu/science/quinn/p53page/basics3.htm`

Les deux brins d'ADN assemblés sont anticomplémentaires - autrement dit une Thymine sur un brin sera toujours en face d'une Adenine et une Guanine fait toujours face à une Cytosine. Une séquence d'ADN est identifiée par un de ses brins comme ATGGAAGTATTTAAAGCGCCACCTATTGGATAAAG. A partir de ce brin, il est possible de déduire le brin anticomplémentaire.

Lorsque les organismes se reproduisent, chaque chromosome doit être dupliqué. Pour réaliser cela, la nature sépare les deux brins et construit deux nouveaux brins pour former deux doubles hélices, grâce à la propriété d'anticomplémentarité de l'ADN.

Toutes les protéines réalisées par un organisme sont encodées dans son ADN. En effet, les protéines sont réalisées dans la cellule à partir de l'information présente dans l'ADN. La figure 1.3 met en évidence trois procédés présents dans la cellule permettant la circulation de l'information :

La réplication est le procédé permettant la copie de l'ADN telle qu'expliquée ci-dessus.

La transcription est le processus qui transforme l'information de l'ADN en ARN par anticomplémentarité dans le noyau. L'ARN est semblable à l'ADN mais il n'est constitué que d'un brin et il peut sortir du noyau de la cellule transportant ainsi l'information de l'ADN.

La traduction est le processus qui transforme l'information de l'ARN en protéine. Le ribosome se place sur l'ARN, lit l'information pour lier les acides aminés qui formeront la protéine. Le lien entre l'ARN et la protéine est établi par le code génétique : à

chaque codon (= triplet de nucléotides) correspond un acide aminé.

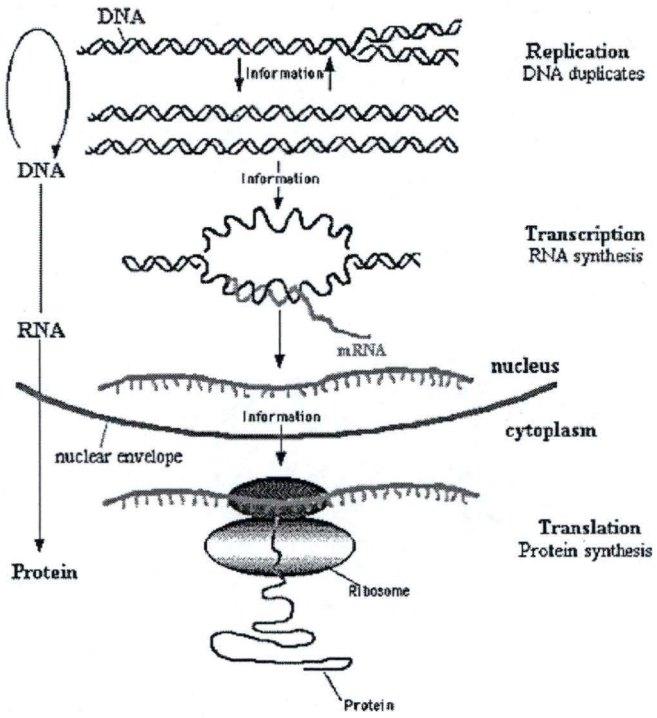


FIG. 1.3 – *Réalisation d'une protéine (transcription et traduction)*
source : `http://www-stat.stanford.edu/~susan/courses/s166/node2.html`

Un gène est une partie d'ADN codant pour une protéine. Or seulement 5% de notre ADN est identifié comme étant des gènes. Les 95% restant ont un rôle encore inconnu. Les biologistes pensent que ces parties ont un rôle de régulation. Cette partie d'ADN est appelé intron[3].

1.2 La bioinformatique : entre informatique et biologie

En laboratoire, à partir des années 50, il fut possible de déterminer la séquence à la fois des protéines et de l'ADN (les techniques sont différentes mais produisent toutes deux une chaîne de caractères représentant la chaîne d'acides aminés ou de nucléotides[4]). Cette expérience, appelé séquencage, est relativement longue, il était donc nécessaire de stocker les résultats pour ne pas devoir la répéter. Dès les années 80, les ordinateurs ont permis le stockage de ces données, c'est le début de la bioinformatique.

3. Intron est une opposition à exon, le nom de la partie codante car elle permet d'exprimer de la nouvelle information.

4. La technique utilisée pour l'ADN est plus rapide car l'ADN n'a que quatre constituants.

Nous pouvons définir la bioinformatique comme :

> Bioinformatics is the application of computer technology to the management and analysis of biological data. The result is that computers are being used to gather, store, analyse and merge biological data. [EBI, 2004a]

Bien plus qu'une application de l'informatique, la bioinformatique est devenue une nouvelle façon de faire de la biologie en menant de nouveaux types d'expériences très peu coûteuses.

Les découvertes réalisées par la bioinformatique modifient la vision actuelle de la biologie. Ces découvertes ont un impact sur la modélisation des données, les outils nécessaires, etc. Elles modifient ainsi également la bioinformatique.

La bioinformatique est donc un secteur de recherche interdisciplinaire entre la biologie et l'informatique. Le but ultime de la bioinformatique est de mieux comprendre le fonctionnement des organismes vivants. Ces connaissances peuvent avoir un impact sur des champs aussi divers que la santé humaine, les entreprises pharmaceutiques, l'agriculture, l'environnement, etc.

1.2.1 Les bases de données

On distingue les bases de données par l'information qu'elles stockent : l'ADN, les protéines, les gènes (ADN codant pour une protéine), les articles scientifiques, les motifs (fragments de protéine qui sont liés à une certaine fonction, une propriété), les structures (formes de la protéine), etc.

On rencontre également des bases de données essayant de faire le lien entre ces différentes données.

La figure 1.4 permet d'avoir une idée du nombre de données aujourd'hui stockées. On trouve principalement de l'ADN[5], puis des protéines. Peu de structures ont été découvertes, récolter plus de données concernant ces structures est un défi pour la bioinformatique.

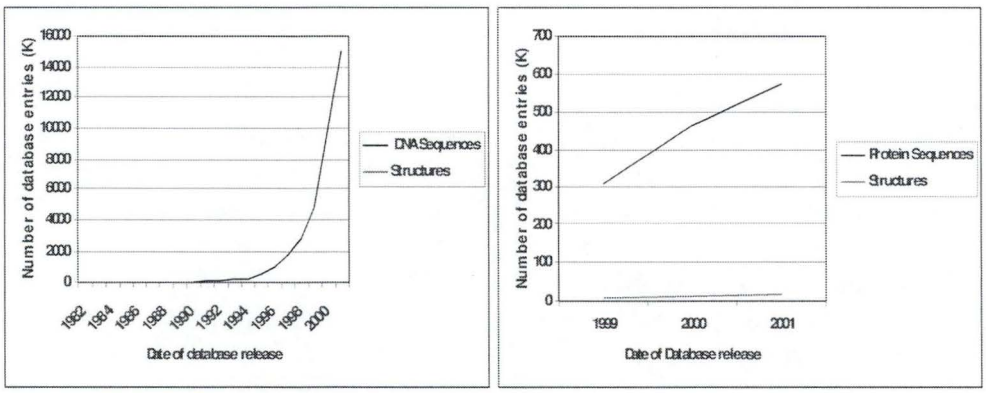


FIG. 1.4 – *Evolution de la taille des bases de données [BEN, 2004b]*

5. Le génome humain est complètement séquencé et stocké dans des bases de données.

Enfin, il est à noter qu'il n'existe pas qu'une seule base de données pour chaque type de données. Ainsi, il existe trois grosses banques d'ADN généralistes : EMBL (gérée par EBI, Europe), GENBANK (NCBI, USA) et DDBJ (Japon). Chaque groupe collecte une partie des découvertes réalisées dans les laboratoires (par expérience traditionnelle ou au moyen de la bioinformatique) et les nouvelles données ou mises à jour sont transmises quotidiennement entre ces bases de données.

Ces bases de données sont généralistes, c'est à dire qu'elles stockent des données provenant de toutes les espèces. On trouve aussi des bases de données spécialisées qui stockent les données ne provenant que d'une seule espèce.

On peut également constater d'autres différences entre les bases de données : comme par exemple : les différents processus de collecte de l'information. Par exemple, une protéine peut être obtenue de deux manières différentes : in silico (déduite à partir de la séquence nucléique, par simple traduction du ou des exons la codant) ou isolée à partir de la cellule. SWISSPROT contient seulement des protéines isolées dans les cellules tandis que trEMBL contient toutes les protéines déduites des séquences nucléiques contenues dans EMBL. La différence est intéressante pour les biologistes car la déduction ne permet d'établir qu'une probabilité et non une certitude sur l'existence de cette protéine dans la nature.

1.2.2 Les programmes

Une masse d'informations considérables est donc mise à la disposition des biologistes. Ces données peuvent être considérées comme un texte vide de sens. Ce sens doit être apporté par des annotations : commentaire du biologiste sur la structure, la fonction et toutes autres informations utiles sur la séquence. Le défi actuel, maintenant que le génome humain est totalement séquencé (et stocké dans des bases de données), est de pouvoir interpréter ce texte.

Pour ce faire des outils ont été développés pour visualiser ces données, pour prédire des propriétés d'une séquence (comme sa fonction, sa structure, etc.) et enfin pour comparer des séquences.

La similarité est un concept inventé par les biologistes pour mettre en évidence un lien entre deux séquences : deux séquences similaires possèdent une séquence ancêtre commune et la différence entre elles se traduit par l'existence de mutations[6], insertions et délétions[7] accumulées au cours de l'évolution. Ces séquences partagent également leur structure ainsi que leur fonction.

Sans ordinateur, pour vérifier qu'une séquence est similaire, il suffit d'étudier si on peut dériver une séquence de l'autre autrement dit retracer l'évolution. Les bioinformaticiens ont créé des outils qui permettent non seulement de comparer des séquences (alignement) mais qui permettent également de rechercher dans les bases de données les séquences les plus similaires à une séquence (recherche de similarité auprès de base de données). L'annexe A fournit des informations sur ces outils.

La recherche de similarité entre deux séquences constitue un outil fondamental de la bioinformatique. Le but est d'identifier la fonction d'une séquence en la comparant à une ou plusieurs autres séquences, notamment au pool de séquences de référence connues et

6. Une mutation est une modification de la séquence par altération de la séquence d'un fragment d'*ADN ou protéine* allant de la modification d'une seule *paire de nucléotides ou acides aminés* au réarrangement ou à la perte d'un morceau.

7. La délétion constitue une suppression d'un acide aminé ou d'un nucléotide.

annotées, stockées dans les banques de données.

D'autres outils ont été réalisés pour analyser les séquences : prédire la protéine encodée par un gène, réaliser le profil hydrophile/hydrophobe d'une protéine, prédire la structure, etc.

Ces outils de prédiction se basent sur des propriétés chimiques ou des méthodes statistiques ou encore d'autres techniques. Le choix de l'outil a donc un impact sur le résultat. Pour éviter de formuler des conclusions hâtives ou erronées, il est conseillé d'utiliser plusieurs d'outils qui vont permettre de confirmer ou d'infirmer une hypothèse.

Enfin, l'informatique apporte des outils performants permettant la visualisation des informations biologiques complexes (par exemple la visualisation de la structure 3D d'une protéine).

1.2.3 Les défis

Nous ne sommes qu'au début de la compréhension du fonctionnement de notre organisme mais la bioinformatique a permis de réaliser des avancées incroyables comme le séquençage et le stockage de tout le génome humain. Désormais, la biologie tente de comprendre le sens des données stockées, pour par exemple comprendre comment l'organisme régule la production de protéines, etc. L'interprétation des données est donc devenue la raison d'être de la bioinformatique.

La connaissance nécessaire pour réaliser une expérience correctement (c'est à dire en utilisant les bons outils et les bonnes valeurs pour les différents paramètres) en bioinformatique est relativement complexe pour un biologiste en laboratoire qui utilise la bioinformatique environ une fois tous les six mois. Une simplification de certains aspects de cette connaissance est nécessaire ainsi qu'une amélioration de l'accès à cette connaissance.

Cette connaissance provient :

- des programmes.
  Il existe énormément de méthodes, d'outils permettant de réaliser une tâche : chaque programme, a été conçu pour s'adapter à une situation précise (taille des données, type de résultat, etc.). D'après la situation, il faut choisir le programme adéquat . Ceci n'est pas toujours facile car l'utilisateur ne connaît pas le programme et/ou n'a pas d'informations suffisantes pour l'aider dans le choix de ce programme.
  Par exemple, pour réaliser un alignement, il existe différentes méthodes qui produisent différents types de résultats comme un fichier mettant en évidence les mutations, insertions/délétions ou encore un graphique mettant en évidence les zones de similarité. Ces différentes méthodes et outils sont expliqués dans l'annexe A.

- des paramètres.
  Les paramètres des programmes ne sont pas que des éléments biologiques. En effet, les programmes sont généralement des formalisations d'un processus naturel. Les paramètres correspondent donc à des nouveaux concepts, généralement créés pour quantifier des phénomènes biologiques comme par exemple, le fait qu'une mutation d'acides aminés se produit si ceux-ci ont des caractéristiques proches[8], le programme

8. La probabilité qu'un acide aminé hydrophile soit remplacé par un acide hydrophobe est très très faible.

se base donc sur une matrice contenant les coûts de mutation des différents acides aminés (appelée matrice de substitution). Les valeurs des paramètres auront un impact sur le résultat.
Comprendre le fonctionnement du programme et des différents concepts permettra d'influencer le résultat selon l'environnement de l'expérience et ainsi obtenir un résultat de meilleure qualité.

- d'autres éléments extérieurs à la tâche comme le format, etc.
Suite à l'évolution de la bioinformatique, de nombreux formats de données cohabitent. Le format est un ensemble de conventions utilisées pour représenter la donnée. Chaque programme n'accepte les paramètres que dans un nombre restreint de formats, généralement le format correspond à un format développé par l'auteur du programme. L'utilisateur doit donc fournir au programme la donnée dans un format accepté et au besoin reformater cette donnée (pour disposer de cette donnée dans un format acceptable) (plus d'informations dans le chapitre 4).

Enfin, l'utilisateur a rarement accès directement aux programmes (sauf au moyen de la ligne de commande) mais plutôt à des services c'est à dire une interface graphique lui permettant d'interagir avec tout ou une partie des fonctionnalités du programme.

Ces services sont accessibles au moyen d'Internet mais peu d'outils permettent de faire le lien entre tous les services disponibles : que ce soit un catalogue de services disponibles ou un outil permettant d'enchaîner aisément différents programmes ou services proposés sur différents sites.

1.3 Une expérience

Pour illustrer notre travail, nous allons utiliser une expérience basique : le biologiste possède une donnée (de type protéine), il a une idée de la fonction de cette protéine mais voudrait des informations complémentaires pour étayer cette thèse. Il va donc chercher des séquences similaires ayant cette fonction qui lui permettront de confirmer son hypothèse.

L'utilisateur a stocké une donnée sur son ordinateur dans un fichier texte au format FASTA. Ce format permet de stocker des séquences c'est à dire une chaîne de caractères reprenant soit des acides aminés, soit des nucléotides. Il se compose d'un symbole > suivi du nom de la séquence et une éventuelle description puis un retour à la ligne suivi de la séquence d'ADN ou de protéine (suite d'acides aminés ou de nucléotides).

L'utilisateur a appelé cette donnée S6A4_MOUSE comme le nom de cette protéine dans la base de données où il a trouvé cette information (voir figure 1.5).

Le biologiste pense que cette donnée sert au transport d'une certaine molécule. Il va donc rechercher des séquences similaires au moyen de BLAST[9]. Ce biologiste connaît l'adresse d'un site offrant un accès à ce service et se rend sur ce site[10].

La figure 1.6 représente une partie de ce formulaire. l'utilisateur peut également choisir

9. BLAST est le logiciel le plus populaire de bioinformatique.

10. L'institution offrant accès à ce service est le nœud suisse de l'EMBnet (*European Molecular Biomedical network* est un groupe de nœuds scientifiques collaborant à travers l'Europe et hors d'Europe).

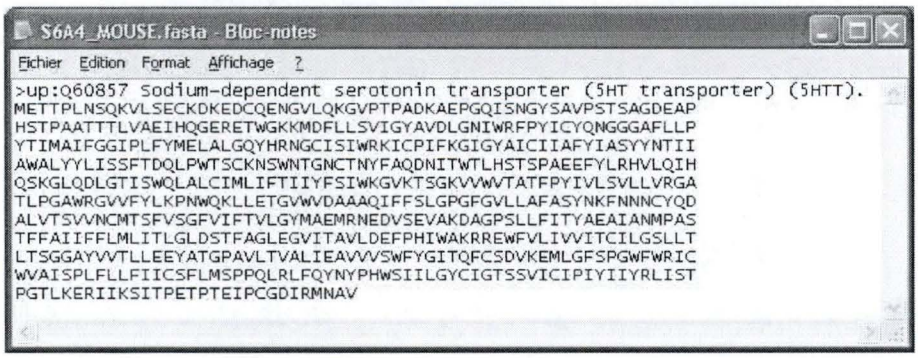


FIG. 1.5 – *Séquence S6A4_MOUSE*

la base de données qu'il veut interroger et fixer d'autres paramètres comme le nombre de données à afficher, le score minimum acceptable, etc.

Vous remarquerez que la donnée présente dans ce formulaire n'est pas exactement identique à celle présente à la figure 1.5. En effet, la séquence stockée est en format *fasta* mais ce programme n'accepte que les séquences en *raw format*. Dans ce cas le changement de format est évident, il suffit d'enlever la première ligne.

La figure 1.7 représente les résultats obtenus. Chaque ligne correspond à une séquence qui est similaire, les informations présentées sont les suivantes : un lien vers la base de données, ainsi que la description de cette donnée, ensuite ce sont les données calculées par le programme : le score de l'alignement, l'E-value[11].

Vous remarquerez que l'utilisateur est intéressé par la séquence S6A2_MOUSE et souhaite stocker cette donnée pour pouvoir travailler dessus.

11. L'E-value calcule la probabilité qu'une séquence non similaire atteigne ce score par hasard.

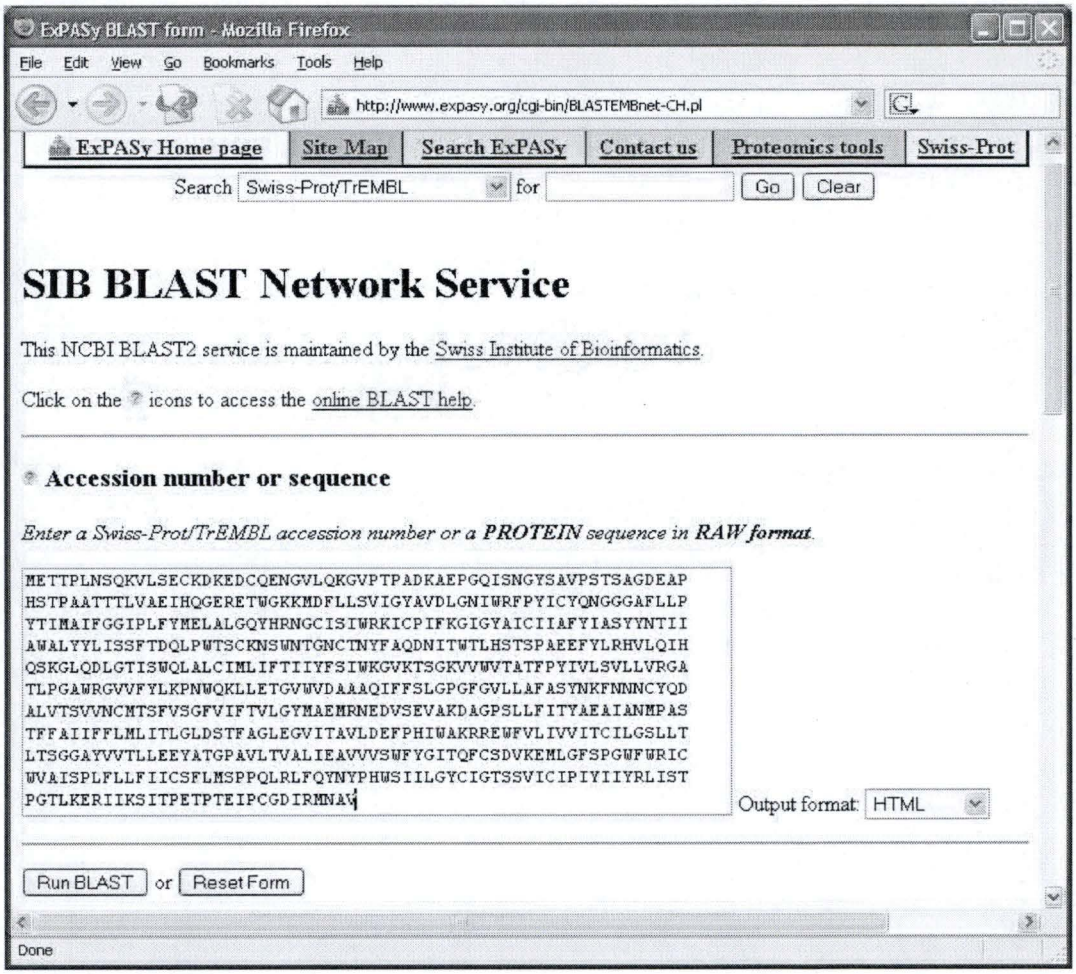


FIG. 1.6 – *Paramétrisation d'un service BLAST*
fournisseur du service : `http://www.expasy.ch/cgi-bin/BLASTEMBnet-CH.pl`

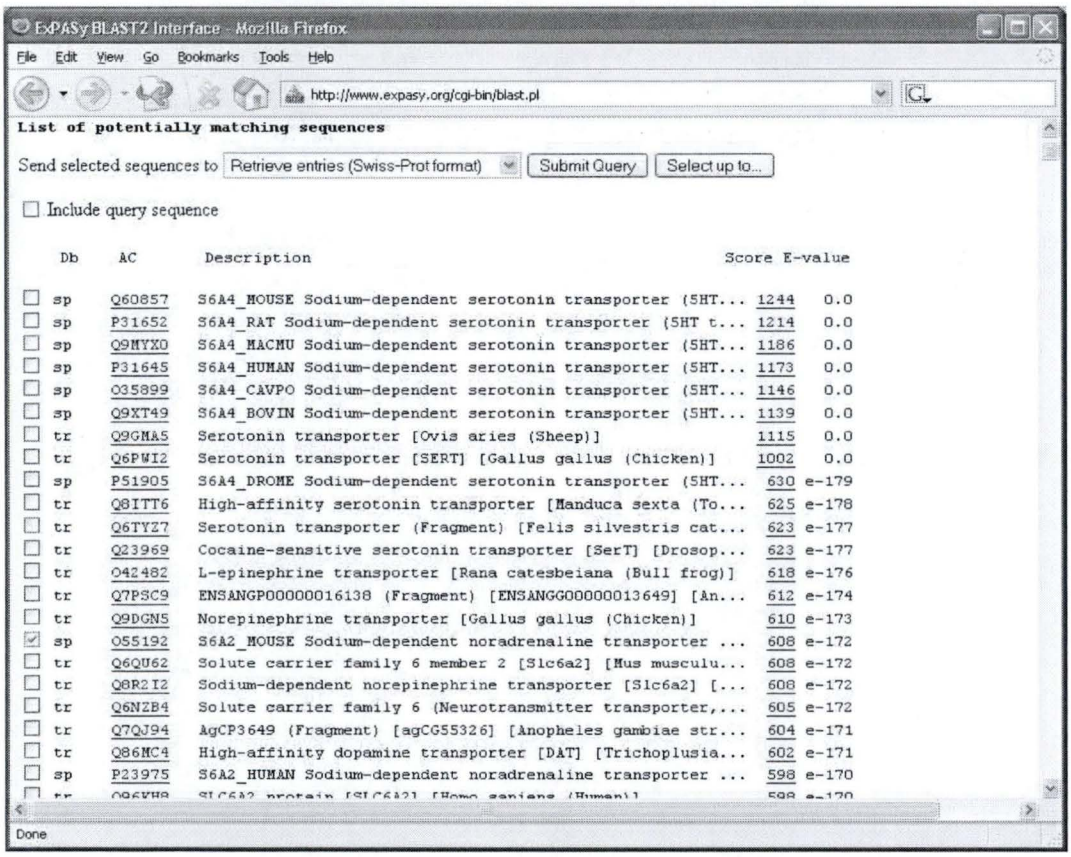


FIG. 1.7 – *Résultat de BLAST*

Ensuite, le biologiste sélectionne certaines données (comme S6A2_MOUSE) qui lui semblent intéressantes et dont il veut avoir plus d'information sur leur similarité.

Divers programmes peuvent l'aider à réaliser cette comparaison entre deux séquences. Il choisit de réaliser un dot-plot et un alignement local (plus d'informations sur ces méthodes dans l'annexe A).

Pour réaliser ces programmes, il doit se rendre sur le site d'un autre fournisseur [12]. Ce fournisseur lui propose l'accès aux deux services. La figure 1.8 représente le formulaire permettant au biologiste de paramétrer le *service alignement local*. Il doit également compléter un formulaire identique pour la paramétrisation du *dot-plot*.

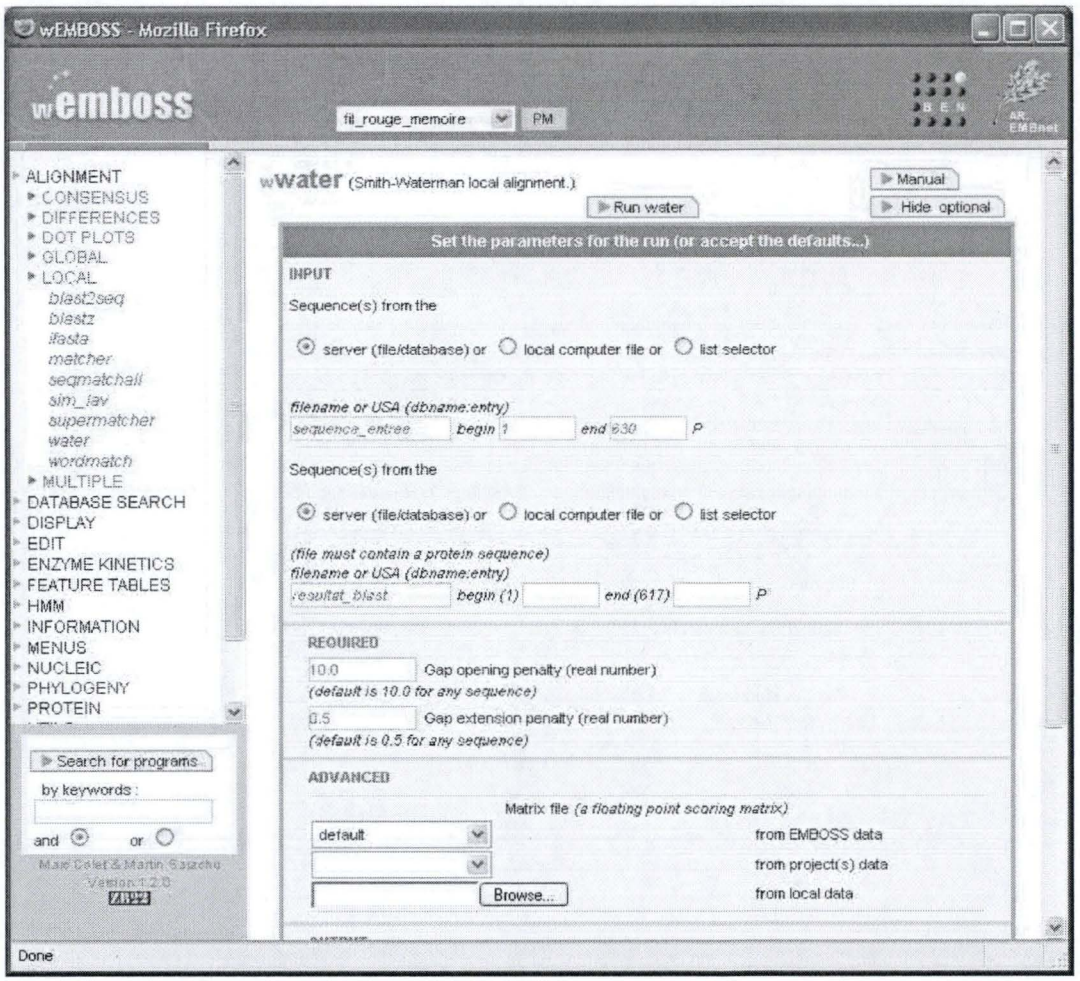


FIG. 1.8 – *Paramétrisation d'un service alignement*
fournisseur du service : `http://www.be.embnet.org/wEMBOSS/`

Le programme d'alignement local réalise une comparaison des deux séquences en essayant de déduire les mutations probables (insertion, délétion, substitution).

12. L'institution offrant accès à ces services est l'IBMM (département de biologie moléculaire de l'ULB) qui est également le nœud belge de l'EMBnet.

La figure 1.9 représente le résultat du programme d'alignement local.

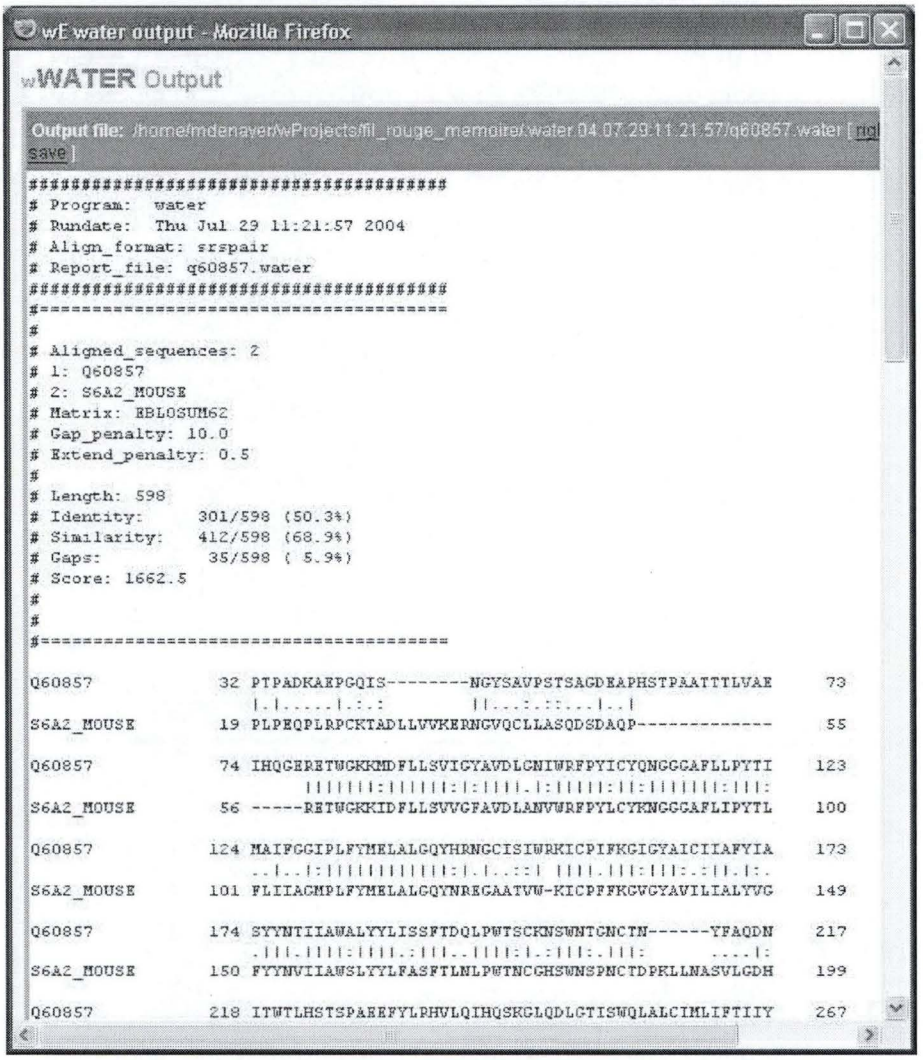


FIG. 1.9 – *Résultat de l'alignement*

Le programme de dot plot réalise un graphe exprimant les régions de similarité entre deux séquences. Les axes correspondent aux séquences et le graphe présente la fonction : $f(x,y) = 1$ si x et y sont similaires, 0 sinon.

La figure 1.9 représente le résultat du programme de dot plot.

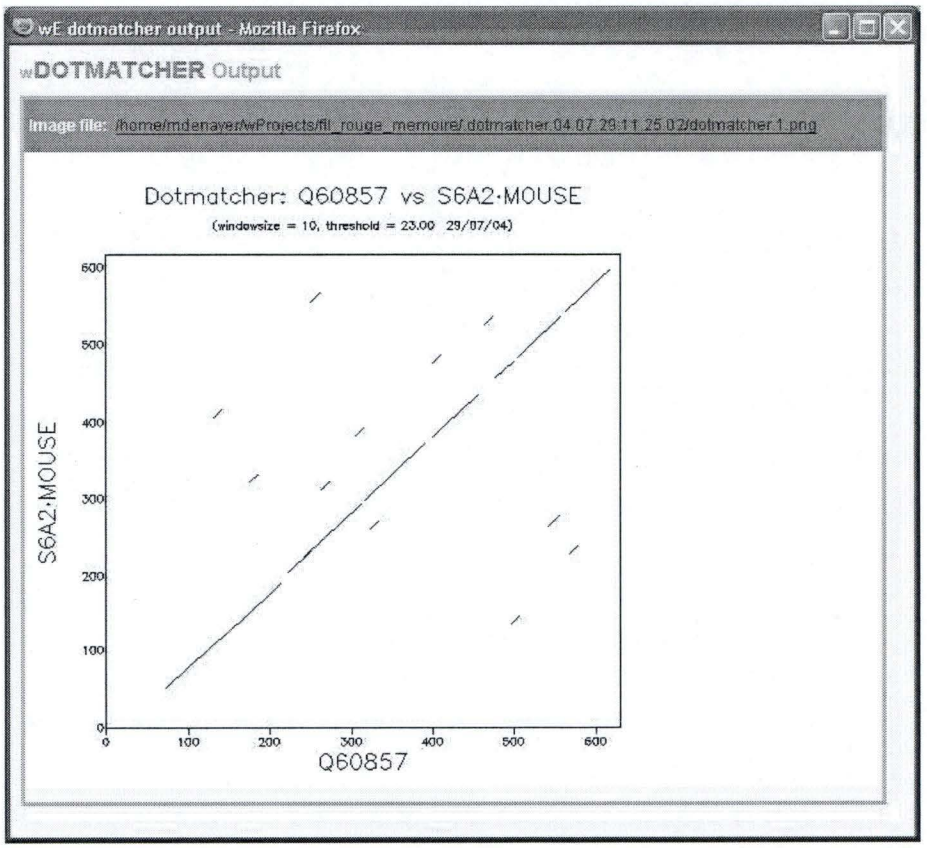


FIG. 1.10 – *Résultat du dot plot*

1.4 Conclusion

La bioinformatique permet aux biologistes de réaliser de nouvelles expériences au moyen de l'informatique. De nombreuses découvertes ont été réalisées par la bioinformatique et ce n'est qu'un début. On peut dire que cette science est en pleine expansion.

Cependant, sa maîtrise par le biologiste n'est pas évidente. En effet, cette science s'appuie sur de nouveaux concepts, peu liés à la biologie. Les bioinformaticiens ont donc de nombreux défis à relever pour aider le biologiste à réaliser au mieux ses expériences.

Chapitre 2

Intérêt d'un système distribué : Bigre

Dans ce chapitre, nous allons mettre en évidence la situation actuelle de la bioinformatique où les différentes bases de données, les différents programmes sont accessibles au moyen d'Internet, pour ensuite mettre en évidence les avantages d'un système distribué. Pour conclure, nous aborderons une initiative BIGRE qui propose justement une solution distribuée adaptée à la bioinformatique.

2.1 Utilisation actuelle de la bioinformatique

Internet a permis le développement de la bioinformatique. En effet, il est devenu plus simple pour les différents utilisateurs d'accéder non seulement aux services et bases de données installés dans son laboratoire mais également d'accéder aux services ou bases de données disponibles ailleurs. De plus de nombreux fournisseurs offrent gratuitement ces différents services. De ce fait, l'offre s'est sensiblement améliorée.

Néanmoins, la gestion est relativement chaotique. En effet, l'utilisateur gère seul, sans outil spécifique ces différents services répartis sur différents sites. La figure 2.1 représente cette architecture.

Ainsi, dans notre exemple du biologiste désirant réaliser une recherche de similarité puis approfondir ces résultats au moyen d'une comparaison, le biologiste doit d'abord se connecter au serveur2 pour réaliser un BLAST (programme 1). BLAST cherchera dans BD2 les séquences similaires à la séquence requête. A partir du résultat de BLAST, l'utilisateur sélectionnera les données intéressantes. Il pourra ensuite stocker ces données sur son ordinateur au moyen d'un programme de traitement de texte. Ensuite, il se connectera au serveur1 pour réaliser les programmes 1 et 2 qui lui permettront de réaliser respectivement un alignement local et un dot plot. De plus, l'utilisateur pourrait également installer certains programmes sur sa machine.

2.1.1 Limitations de cette architecture

Aujourd'hui, les programmes bioinformatiques sont généralement capables de coopérer avec d'autres programmes amis car ils utilisent le même format,etc. Généralement, ces programmes sont regroupés en package et se trouvent sur un même site (comme les programmes

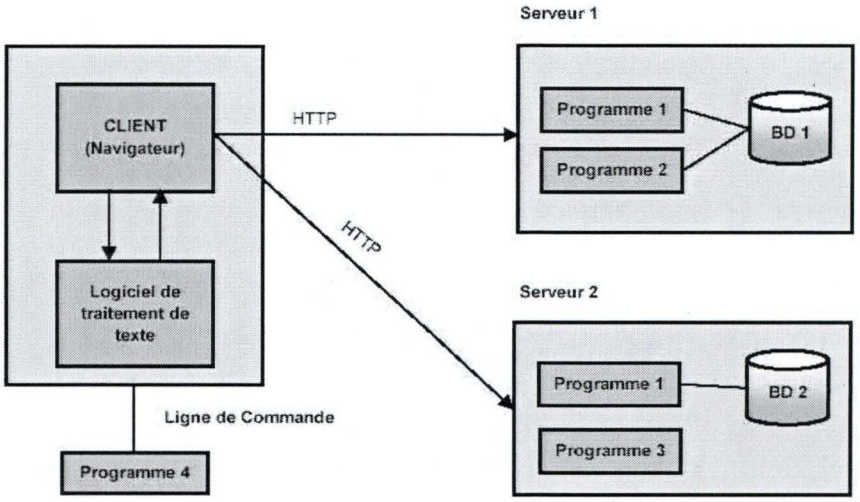


FIG. 2.1 – *Architecture actuelle [Debaisieux et Desouza, 2003]*

utilisés dans notre expérience pour réaliser l'alignement et le dot plot).

Si le biologiste veut faire interagir ces programmes avec d'autres programmes, cela devient difficile car il doit réaliser des opérations superflues : des changements de site, des opérations de reformatage.

Les sites des laboratoires très connus sont fort utilisés. Les requêtes des différents utilisateurs sont donc ralenties par les requêtes des autres utilisateurs.

Pour améliorer la disponibilité du service, ces laboratoires ont mis en place des sites miroirs. Malheureusement, peu de biologistes les utilisent car ils ne connaissent pas ou ne retiennent pas les adresses de ces sites.

Dans l'architecture présentée ci-dessus, le site 1 a accès à une base de données. Les programmes n'ont pas besoin d'accéder directement à ces données mais si l'utilisateur fournit comme paramètre en entrée un identifiant de cette base de données, le programme doit pouvoir faire le lien avec la valeur de cette donnée. Cette raison explique la présence d'une base de données sur ce site.

Tous les fournisseurs doivent donc avoir accès aux bases de données (sauf s'il interdisent ce type de valeur). Aujourd'hui, les fournisseurs répliquent donc quotidiennement ces bases de données pour que les différents fournisseurs disposent toujours de l'information la plus récente.

L'utilisation d'internet pose également un problème en terme de sécurité. En effet, dans certains cas, les données doivent rester confidentielles. Or, aujourd'hui, l'envoi de donnée pour les transmettre à un programme n'est pas sécurisé, de même que l'envoi de résultat. De plus, rien ne permet à l'utilisateur de savoir si le programme n'enregistre pas ses données sur le site du fournisseur.

Les biologistes qui désirent conserver la confidentialité de leurs données n'utilisent donc que des programmes installés dans un intranet sûr.

2.1.2 Dialogue utilisateur-service

Modéliser le comportement de l'utilisateur permet de mettre en évidence les difficultés qu'il rencontre lorsqu'il réalise des expériences au moyen de l'ordinateur. Nous avons donc repris l'exemple précédent en le modélisant au moyen de *uses cases*. Nous avons divisé l'exemple en deux étapes (voir figures 2.2 et 2.3) : réalisation d'un blast puis sélection des résultats et réalisation d'un alignement ainsi que du dot plot. Chacune de ces étapes est réalisable individuellement mais les enchaîner a un intérêt certain.

UC1 : « Réalisation d'un blast »

Description : Le biologiste désire retrouver les données parentes à une séquence qu'il possède.
Pré-conditions : Le biologiste possède la séquence au format nécessaire au logiciel qu'il désire utiliser. Le biologiste connaît un système permettant d'effectuer sa requête.

Biologiste	**Système**
Le biologiste sélectionne un système (le système est identifié par une adresse web).	
	Le système renvoie les logiciels disponibles. La présentation de cette liste lui est propre.
Le biologiste sélectionne le logiciel BLAST.	
	Le système renvoie un formulaire permettant de paramétrer ce logiciel (le formulaire est propre au fournisseur, il ne propose donc que les bases de données auxquelles il a accès).
Le biologiste fournit au logiciel une séquence et fixe éventuellement les autres paramètres (comme la base de données qu'il souhaite interroger). Puis il demande l'exécution du logiciel.	
	Le système renvoie le résultat de l'exécution de BLAST [1].
Le biologiste sélectionne les données qui lui semblent intéressantes et demande au système de les lui fournir dans un certain format.	
	Le système présente les données selon le format désiré.

Post-condition : Le système a renvoyé la liste des données présentant des similitudes avec la donnée du biologiste. L'utilisateur dispose du résultat de BLAST ainsi que de la liste de séquences intéressantes.

FIG. 2.2 – *UC 1: réalisation de BLAST*

1. Ce résultat se présente sous la forme d'une liste de données présentant des similitudes avec la donnée en entrée ainsi que des compléments d'informations sur la similitude entre chacune des données et la donnée en entrée.

UC2 : « Alignement »

Description : Le biologiste désire disposer d'informations sur la similarité de deux séquences.
Pré-conditions : Le biologiste possède les séquences au format nécessaire aux logiciels qu'il désire utiliser. Le biologiste connaît un système offrant un accès à différents services d'alignement.

Biologiste	**Système**
Le biologiste sélectionne un système (le système est identifié par une adresse web).	
	Le système renvoie les logiciels disponibles. La présentation de cette liste lui est propre.
Le biologiste sélectionne le logiciel nécessaire pour réaliser un dot plot et lui fournit les deux séquences à aligner (et éventuellement fixe les autres paramètres).	
	Le système renvoie le graphique (dot plot) présentant les régions de similarité.
Le biologiste sélectionne le logiciel nécessaire pour aligner des séquences et lui fournit les deux séquences à aligner (et éventuellement fixe les autres paramètres).	
	Le système renvoie les deux séquences alignées.
Le biologiste interprète ces résultats.	

Post-condition : L'utilisateur dispose d'un graphique et d'un alignement de ces deux séquences expliquant leur lien de similarité.

FIG. 2.3 – *UC 2: réalisation d'un alignement*

On constate que l'utilisateur doit connaître les informations suivantes :

- les systèmes qui offrent accès aux logiciels, aux bases de données auxquels il souhaite accéder ;
- le format que les données doivent prendre et au besoin, connaître les différents programmes de reformatage auxquels il devra faire appel.

Or ces informations ne sont pas en lien direct avec le travail du biologiste. Ces informations ne devraient donc pas intervenir dans ce travail pour ne pas être un frein à l'utilisation de la bioinformatique.

2.2 Apport d'un système distribué

2.2.1 Système distribué

On peut définir un système distribué comme :

"Un système distribué est une collection d'hôtes autonomes qui sont connectés par l'intermédiaire d'un réseau. Chaque hôte héberge et exécute un ou plusieurs composants tout en supportant un middleware, qui permet aux composants du système de coordonner leurs activités d'une telle façon que les utilisateurs perçoivent le système comme une capacité de traitement unique et intégrée. On oppose généralement système distribué à système centralisé." [Englebert, 2003]

Il va permettre de gérer les exigences suivantes :

- évolution
  - quantitative : montée en charge
  - qualitative : changement de langage, changement de la conception de l'architecture
- hétérogénéité : utilisation conjointe de composant développés from-scratch, de composant achetés et d'élément pré-existant (legacy).
- Partage et accès aux ressources (par exemple des données, des programmes). Remarque : lors du partage, il y a également un contrôle des accès à ces ressources.

2.2.2 Système distribué et bioinformatique

Les mémoires [Debaisieux et Desouza, 2003, Buyle et Dallons, 2003] ont menés à la réalisation d'un prototype de système distribué adapté à la bioinformatique.

Les apports d'un système distribué sont évidents :

" Leur utilisation (technologie de l'information et de la communication) dans le cadre de la problématique de la bioinformatique assurerait une augmentation de la sécurité, une réduction de la réplication systématique des outils et des sources de données mais aussi une amélioration des conditions d'utilisation de ces outils pour le biologiste." [Buyle et Dallons, 2003]

Enfin, de nouveaux services pourront se greffer à ce système : un outil permettant l'enchaînement de service, etc.

2.2.3 Dialogue utilisateur-service

Le système distribué permettra à l'utilisateur d'interagir directement avec tous les fournisseurs au moyen un seul point d'accès. On peut représenter le comportement de l'utilisateur et du système distribué au moyen de la figure 2.4. Dans ce *use case*, l'utilisateur réalise toujours le même travail : réalisation d'un blast et approfondissement de certains résultats.

UC Simplifié : « Recherche de séquences similaires »

Description : le biologiste désire obtenir des informations sur les séquences similaires à une de ses séquences.
Pré-condition : La donnée est déjà stockée par l'utilisateur.

Biologiste	**Système**
Le biologiste se connecte au système.	
Le biologiste sélectionne le service BLAST.	
	Le système affiche le formulaire permettant de paramétrer ce service.
Le biologiste introduit sa séquence (ou un lien) ainsi qu'éventuellement d'autres paramètres comme la base de données qu'il souhaite interroger.	
	Le système localise le service disposant d'un accès à la base de données (par défaut SWISS-PROT).
	Le système effectue la requête et renvoie les résultats.
Le biologiste sélectionne les enregistrements dont il voudrait plus d'informations et soumet la demande de comparaison entre ces données et la donnée en entrée au système.	
	Le système affiche le formulaire du service dot plot et du service alignement.
Le biologiste fixe éventuellement les paramètres (autres que séquences en entrée).	
	Le système localise le service alignement et le service dot plot.
	Le système effectue les deux requêtes et renvoie les deux résultats (alignement et dot plot) au biologiste.
Le biologiste interprète ces résultats.	

Post-condition : Le biologiste a pu infirmer ou non son hypothèse. De plus, il dispose des différents résultats : BLAST, alignement, dot plot.

FIG. 2.4 – *UC simplifié: réalisation d'une recherche par similarité accompagnée d'un approfondissement de la similarité de certaines séquences*

2.3 Architecture de Bigre

Le système BIGRE[IBMM/ULB *et al.*, 2004] présente une architecture composée de trois types d'éléments : les clients, les médiateurs et les services. La figure 2.5 représente cette architecture.

Le **client** assure les interactions entre les utilisateurs et les différents services de manière homogène et cohérente. Ainsi, l'utilisateur utilisera une interface unique et adaptée à son

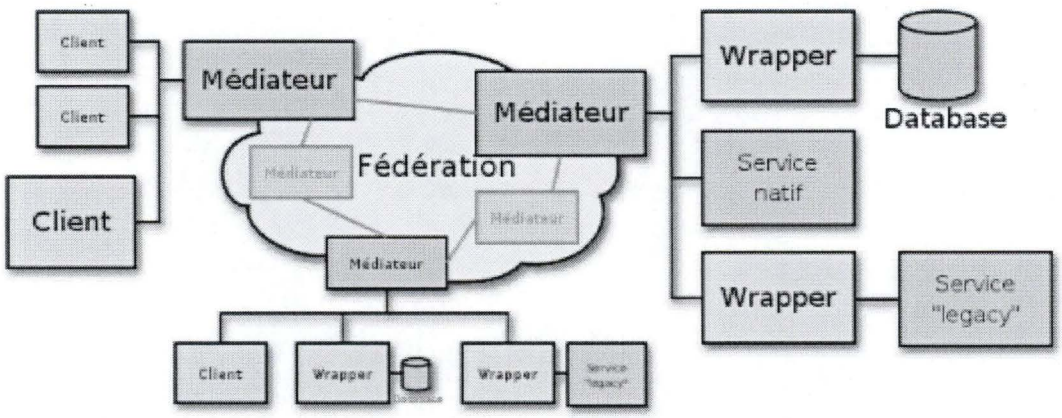


FIG. 2.5 – *Architecture de BIGRE [Buyle et al., 2004]*

profil, indépendamment de la nature des services utilisés, de leur implantation, des politiques de sécurité, etc.

Les **services** sont soit des composants "wrapper" intégrant des outils legacy ou des bases de données, soit des services natifs, c'est à dire des applications à valeur ajoutée (traitements, extraction, etc.). Tous les services présentent une même et unique interface directement utilisable par les médiateurs et les clients.

Les **médiateurs** fournissent les fonctionnalités assurant la mise en relation des clients et des services. Ils permettent, entre autre, la découverte dynamique de services, la sécurisation des communications et des accès, et enfin l'intégration sémantique des services sur base d'une ontologie commune des services et des données. Ces médiateurs sont répartis dans une fédération afin de pouvoir déléguer leur gestion à des entités scientifiques ou économiques différentes. Leur conception est elle-même distribuée afin de pouvoir assumer une montée en charge.

2.4 Conclusion

Ce chapitre montre la difficulté pour le biologiste de réaliser un enchaînement de services. En effet, l'usage actuel de la bioinformatique nécessite une connaissance "extérieure à la bioinformatique" comme les nombreux formats.

Nous avons ensuite analysé les opportunités offertes par un système distribué. En effet, celui-ci permet de gérer ces contraintes de sorte que le biologiste puisse se concentrer sur son expérience. Enfin, nous avons abordé l'architecture utilisée dans le projet BIGRE dont l'objectif est de réaliser un système distribué dans le cadre de la bioinformatique.

Deuxième partie

Analyse

Chapitre 3

Vue générale

Dans le chapitre précédent, nous avons montré les avantages d'un système distribué de services bioinformatiques pour l'utilisateur et le fournisseur. Cependant, ces avantages peuvent être multipliés par une modélisation intelligente des données. En effet, nous allons montrer dans ce chapitre que l'interface du client est peu structurée et beaucoup trop hétérogène. Ensuite, nous proposerons une modélisation structurant les diverses informations, permettant la recherche et l'exécution de services.

Ces informations permettront également de modéliser les divers fichiers reprenant l'information stockée sur les médiateurs comme le catalogue de services disponibles.

3.1 Utilisation actuelle

La figure 3.1 représente l'utilisation actuelle de la bioinformatique. L'utilisateur a accès à des services réalisés par des auteurs par le biais des fournisseurs parfois moyennant finances. Les éléments en bleu sont les acteurs et ceux en vert sont les éléments visibles par l'utilisateur.

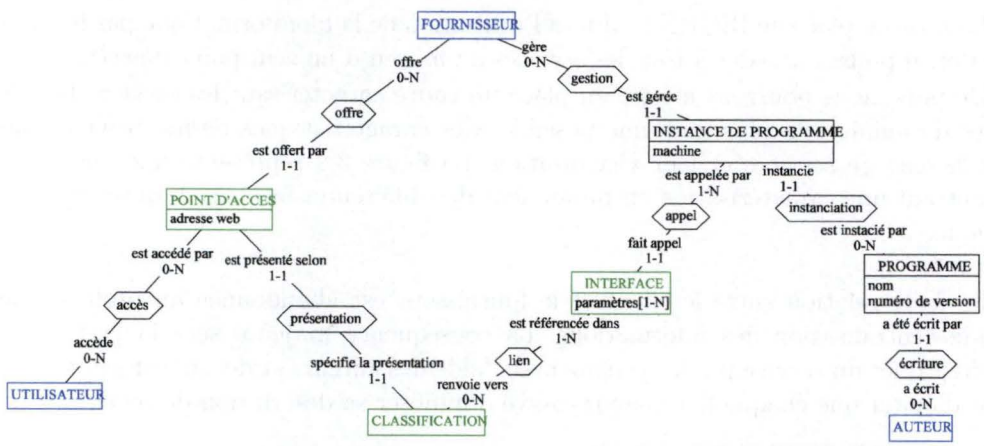


FIG. 3.1 – *Modélisation de la situation actuelle*

Internet permet aujourd'hui à l'utilisateur d'accéder à de nombreux services offerts par les fournisseurs mais il reste nécessaire de connaître l'adresse web du point d'accès, voire du service. Généralement, l'utilisateur utilise ses favoris et donc, n'accède qu'à un petit nombre de points d'accès.

De plus, les interfaces se référencent peu entre-elles, ce qui ne permet pas à l'utilisateur de découvrir de nouveaux services, de trouver un fournisseur offrant des interfaces soit plus simples, soit plus compliquées ou offrant un accès à un programme déterminé.

Le lien entre les auteurs des programmes est assuré par les fournisseurs de services. Ce lien est très variable, il dépend en effet de la politique des différents fournisseurs : certains ont une approche simplificatrice du problème pour favoriser l'utilisation par les utilisateurs débutants, tandis que d'autres présentent tous les paramètres pour permettre aux experts une utilisation la plus précise du service. L'apparence d'un même programme peut donc varier d'un fournisseur à l'autre.

De plus, le fournisseur gère seul son point d'accès et l'organise comme il souhaite. On constate des divergences dans les classifications utilisées par les différents fournisseurs.

Le fournisseur ne transmet pas ou peu d'informations sur le programme utilisé : en général, une simple référence au nom. Or, ces informations, souvent incompréhensible par les biologistes, permettent d'assurer la ré-exécution d'une expérience dans les mêmes conditions[1], chose essentielle en science.

Enfin, il est parfois intéressant d'enchaîner des services, d'enregistrer un appel[2] afin de le ré-exécuter. Or seuls des scripts, c'est à dire des programmes écrits en Perl par le biologiste ou éventuellement fournis par le fournisseur de service, permettent de réaliser ces actions. De plus, ces scripts sont impossibles à maîtriser sans une certaine connaissance en programmation.

3.2 Situation idéale

La mise en place de BIGRE facilitera l'utilisation de la bioinformatique par le biologiste. En effet, il pourra accéder à tous les services au moyen d'un seul point d'accès.

De plus, nous pourrons mettre en place un cadre caractérisant les services bioinformatiques de manière la plus uniforme possible. Ces caractéristiques permettront d'améliorer la recherche de services et leur visualisation. La figure 3.2 représente une vue du service permettant une caractérisation en profondeur des différentes facettes d'un service bioinformatique.

Le forte relation entre le client et le fournisseur est abandonnée au profit d'une plus grande centralisation des informations. La conséquence majeure sera la gestion des caractéristiques du service par le système avec l'aide des auteurs et des fournisseurs ; l'objectif étant d'éviter que chaque fournisseur essaye d'imposer sa description de service.

1. Un changement de version est une explication si le programme ne renvoie plus les mêmes résultats avec les mêmes données en entrées.

2. Les informations sur l'appel sont les informations sur le service ainsi que les valeurs des différents paramètres.

De plus, lors de l'exécution de services, le client n'est plus tenu de connaître le nom du fournisseur lui offrant accès aux services. Toutefois, l'utilisateur peut toujours être lié à certains fournisseurs par un contrat qui lui offre accès à certains services.

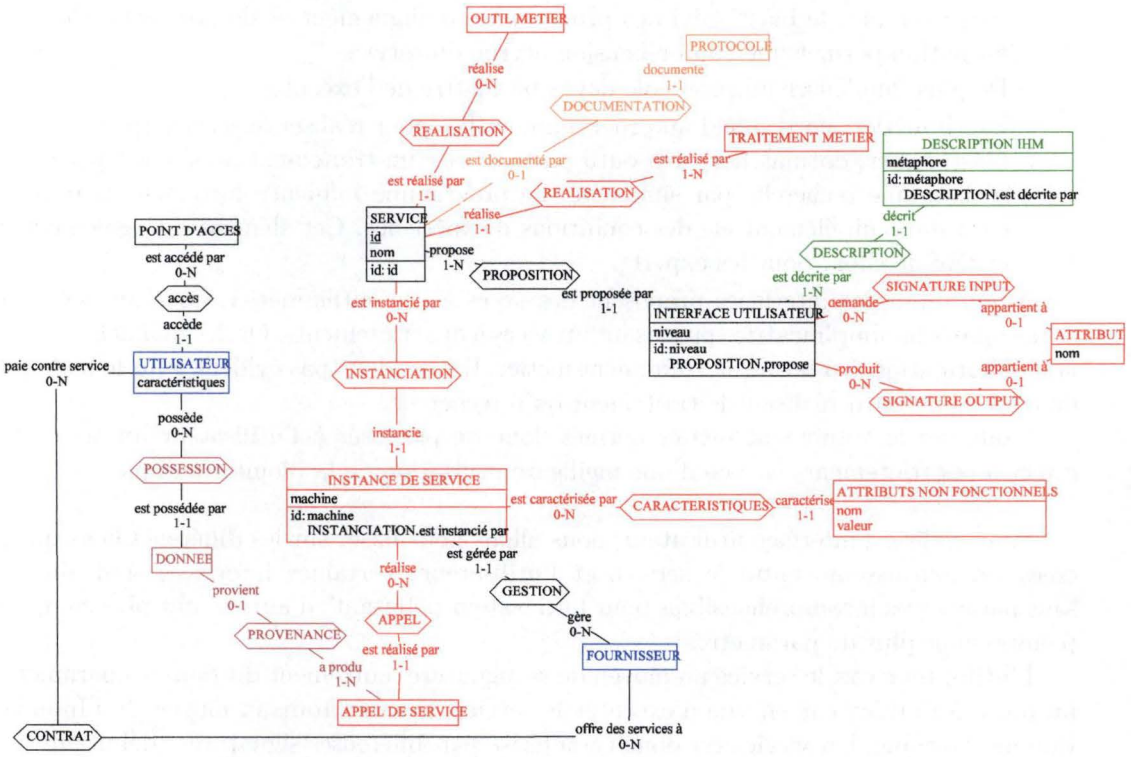


FIG. 3.2 – *Modélisation idéale d'un service*

Le système doit d'abord prendre en compte le fait que l'utilisateur possède des données qui proviennent généralement de l'exécution d'un service. Intégrer ces données permettra une meilleure gestion par le biologiste et des améliorations considérables dans la bioinformatique[3].

Ensuite, nous avons voulu mettre en évidence deux vues possibles du service : la vue générale et une vue utilisateur. La première vue reprend les les caractéristiques générales. La seconde décrit les attentes de l'utilisateur.

Un service est caractérisé par une vue générale et plusieurs vues utilisateur. En effet, les attentes des utilisateurs peuvent être différentes d'après leur niveau de connaissance,etc.

La vue générale décrit le service en termes généraux. Ces caractéristiques sont inhérentes aux services car elles proviennent du code exécuté pour fournir le service. On y retrouve des informations concernant :

– le traitement métier : représente l'action réalisée par le programme (par exemple :

3. Les programmes auront à leur disposition plus d'informations sur la donnée, ce qui leur permettra de fournir des résultats plus précis.

recherche de similarité, alignement). Cette action est évidemment le premier critère dans le choix d'un service. Aujourd'hui, ce critère est régulièrement employé dans les classifications des fournisseurs.

- le protocole : définit l'enchaînement des opérations dans le cas d'un service complexe (par exemple : le blast suivi des programmes d'alignement et de dot plot). Cette information permet une compréhension accrue du service.
  De plus, modéliser un protocole devra permettre de l'exécuter.
- l'outil métier : correspond au programme utilisé pour réaliser le service (par exemple : blast, water, dotmatcher). Un outil particularise un traitement ainsi blast permet de réaliser une recherche par similarité. Le programme influence fortement le résultat, c'est donc un élément clé des conditions d'expérience. Cet élément est également un critère de choix pour les experts.

Aujourd'hui, les interfaces proposent des accès à des outils métier, donc au détriment d'une approche simplificatrice proposant un accès à des traitements. Or, les manuels décrivent la bioinformatique en terme de traitement métier. Il n'est donc pas évident pour le biologiste de trouver un outil réalisant le traitement qu'il recherche.

Modéliser le traitement métier permet donc de proposer à l'utilisateur un accès plus direct à ces traitements en vue d'une meilleure utilisation de la bioinformatique.

Pour réaliser l'interface utilisateur, nous allons nous baser sur les différents liens qu'ont créés les fournisseurs entre le service et l'utilisateur : certaines interfaces sont simples, sans paramètres incompréhensibles pour l'utilisateur débutant ; d'autres sont plus complexe (comprenant plus de paramètres).

L'utilisateur voit le service au moyen de sa signature, autrement dit dans les paramètres proposés à l'utilisateur en vue d'exécuter le service, mais surtout au moyen de l'Interface Homme Machine. Un service est donc caractérisé par différentes signatures : débutant, normal, expert. Chaque signature peut être vue au moyen de différentes IHMs pour permettre à chaque utilisateur de conserver la présentation à laquelle il est habitué.

Les habitudes de l'utilisateur (attribut `caractéristique`) font le lien avec une signature et une IHM pour chaque service de sorte que le système soit capable de fournir à l'utilisateur une apparence du service la plus adaptée à ses besoins, ses connaissances.

L'instance de service représente le service accessible sur une machine. Ces instances représentent l'offre du système à ses différents utilisateurs. Ces instances ont des caractéristiques (attribut non fonctionnel) permettant de les différencier comme le coût, la performance, la version du service.

3.3 Conclusion

Ce chapitre présente d'abord l'interface actuelle. Il met en évidence son peu de structuration et son hétérogénéité. Cette hétérogénéité est parfois un frein à l'utilisation de la bioinformatique ; par exemple, il est difficile d'enchaîner des services.

Ensuite, nous proposons une caractérisation du service qui serait partagée par tous les fournisseurs. Cette modélisation est plus complète et permet également une multiple caractérisation de certains aspects du service comme l'IHM, la signature. Cette modélisation permettrait de simplifier l'usage de la bioinformatique.

Les chapitres suivants vont nous permettre d'explorer les différents concepts abordés dans notre modélisation (voir figure 3.2). Ceux-ci ont été répartis en quatre catégories (la couleur sur le schéma indique leur catégorie) : les données (dans la couleur bordeau), le service (en rouge), l'Interface Homme Machine (en vert) et enfin le protocole (en orange).

Un chapitre étudiera en détail les concepts d'une catégorie. Ainsi le chapitre suivant détaillera les données et ainsi de suite.

Chapitre 4

Modélisation des données

La collecte et la découverte d'informations sont les raisons d'être de la bioinformatique. Or l'information est principalement fournie par les données qui proviennent d'un laboratoire ou d'outils bioinformatiques. Les services se basent également sur des données pour créer de nouvelles données. Il est donc indispensable de modéliser ces dernières.

Ces données représentent des informations biologiques comme des protéines mais aussi des données plus classiques comme des entiers, etc. On peut distinguer en deux types : les données détenues par l'utilisateur ou données personnelles et les données mises à disposition par un fournisseur de service. Ces données seront appelées ressources.

Nous étudierons d'abord les données personnelles pour ensuite étudier les ressources. Nous terminerons ce chapitre par un exemple.

4.1 Donnée(s) personnelle(s)

La figure 4.1 (page 36) représente la modélisation des données personnelles.

L'utilisateur dispose d'un espace de stockage qui lui permet d'enregistrer les informations qui lui sont utiles : données biologiques, graphiques, etc. Ces `données` sont caractérisées par un nom donné par le biologiste, ainsi qu'un emplacement dans son espace de stockage.

Comme toute donnée, la donnée personnelle est constituée d'informations. On peut également la caractériser par un type, une provenance générale, etc. Ces propriétés appelées `méta-données` dans le schéma réaliseront une caractérisation permettant, par exemple, une recherche sur base d'une propriété.

A partir des types, on peut établir une hiérarchie. Ainsi on peut dire qu'un entier est un réel (un entier est un sous-type de réel) ou encore une protéine est une séquence (une protéine est un sous-type de séquence).

Une donnée biologique contient de l'information : les différents caractères du fichier, le biologiste va pouvoir interpréter ce texte. Les annotations comprennent cette interprétation. Celle-ci est réalisée par le biologiste à partir :

- de la provenance de sa donnée. La donnée peut provenir d'une expérience en laboratoire, d'une base de donnée. La donnée peut également être le résultat d'une exécution d'un service.

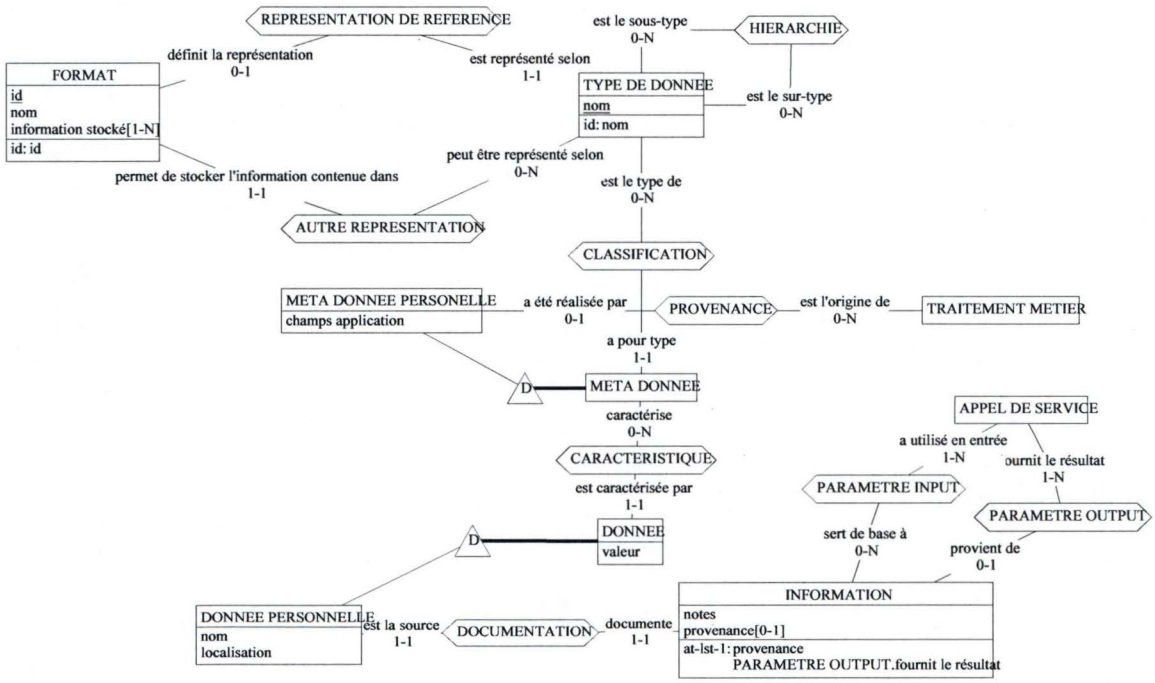


FIG. 4.1 – *Modélisation d'une donnée personnelle*

La provenance établit la base de l'information connue sur la donnée.

- de tous les services mis à sa disposition comme un service de prédiction de structure, un service de recherche de mutation entre deux séquences, etc.
  Ces services vont fournir des informations sur la donnée en produisant une nouvelle donnée (soit la donnée elle-même est porteuse d'information, soit l'interprétation de cette donnée fournira au biologiste cette information).
- des consultations auprès de différentes bases de données :
  les bases de données stockent l'information sur les données biologiques que des scientifiques ont obtenue dans des expérience en laboratoire. Le biologiste pourra trouver dans ces bases de données des annotations qui pourront sous certaines conditions s'appliquer à sa donnée.
- de ses propres déductions faites à partir de ses observations, etc.

Exemple : Nous pourrions modéliser la donnée S6A4_MOUSE (voir figure 1.5 page 13) par la figure 4.2.

Nous n'avions que peu d'informations sur la manière dont le biologiste a obtenu cette donnée. Or, cette modélisation prend en compte des informations qui sont peu voire pas exploitées par les outils actuels: la provenance de la donnée et des informations fournies par le biologiste.

Les méta-données permettent ainsi de stocker le type de la donnée : protéine, ainsi que le traitement qui a permis au biologiste de détenir cette information : une recherche sur base de mots-clés auprès d'une base de données.

Les informations permettent à l'utilisateur de stocker son hypothèse.

La provenance de la donnée permet de connaître les circonstances exactes de sa création. Cette donnée provient d'une recherche effectuée sur base des mots : souris et transporteur. Son hypothèse présentée dans le schéma est donc vraie : cette donnée a pour fonction le transport.

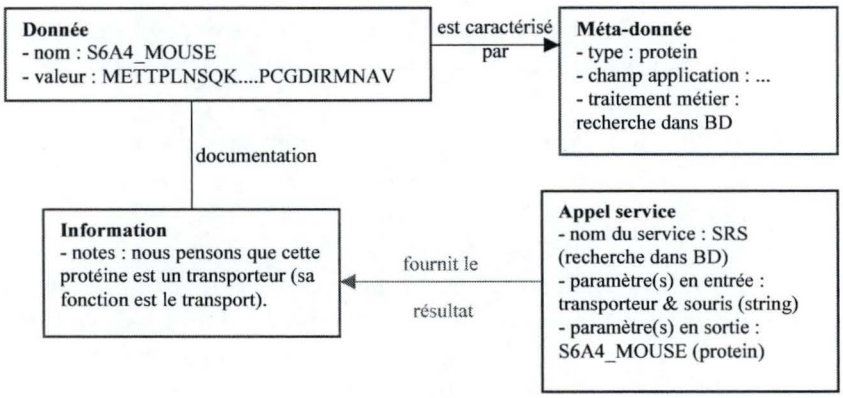


FIG. 4.2 – *Modélisation de la donnée S6A4_MOUSE (application)*

Grâce à cette modélisation de nouvelles informations ont été stockées comme la provenance de la donnée, les hypothèses du biologiste.

4.1.1 Le type prévaut

Le biologiste est capable de caractériser une donnée sur base de son type, autrement dit identifier la sémantique de cette donnée. Ainsi, la donnée S6A4_MOUSE est une protéine.

Or cette caractérisation n'est pas suffisante aujourd'hui car le bioinformaticien devrait également connaître le format de sa donnée. Le format correspond à la syntaxe du document, il précise l'espace occupé par la donnée [1], l'emplacement des différents éléments (nom, annotations, etc.). Connaître le format est indispensable pour pouvoir identifier les services acceptant cette donnée.

Pour chaque type, il existe de nombreux formats. Par contre, le format, par l'information qu'il exprime, fait le lien avec la sémantique de la donnée.

Chaque format a des caractéristiques propres : ils sont simples à écrire par l'utilisateur, ils permettent d'écrire des commentaires, certains sont illisibles par l'humain, etc.

De plus, les formats ne permettent pas d'encoder toutes les caractéristiques d'une donnée ainsi, le *raw format* (voir figure 1.6 page 14) ne permet d'encoder que la séquence [2], tandis que le *format fasta* (voir figure 1.5 page 13) permet d'encoder en plus le nom. Il existe également de nombreux formats permettant de stocker toutes les informations disponibles (fonction, etc.) sur ces données.

Comme le biologiste comprend peu ce concept de format et que ce concept n'a aucune influence sur son travail, il doit avoir l'impression que sa donnée est traitée, visualisée selon

1. L'espace occupé par un entier est, en général, de 4 bytes.
2. Une séquence est une chaîne de caractères correspondant à une protéine ou un bout d'ADN.

son type et pas selon son format. Il doit donc avoir l'impression que les services acceptent désormais tous les formats.

D'autant que cette multitude de formats est non seulement gênante pour le biologiste, mais aussi pour l'informaticien car, si différents services utilisent différents formats, il faudra intercaler des opérations de reformatage afin de pouvoir enchaîner ces services.

La définition d'une norme apparaît donc comme une nécessité. Certains formats ont déjà été développés pour faciliter le transport des données comme la norme BSA et le langage BSML :

La norme BSA[OMG, 2004] formalise les données bioinformatiques au moyen du langage UML [3] pour ensuite les proposer dans le langage IDL [4] et permettre ainsi la réalisation d'objets transitant par le bus CORBA [5]. Différents prototypes s'appuient sur cette norme [Debaisieux et Desouza, 2003, Buyle et Dallons, 2003].

Le langage BSML[LabBook, 2004] est un format XML [6] qui permet l'intégration à la fois des informations propres aux données biologiques (comme le nom, la chaîne de caractères pour la séquence) mais également de l'information annexe comme par exemple un graphique résultant d'un appel de service, etc. Ce format complet n'est plus lisible, ni même éditable par l'humain mais le projet a également développé un outil permettant la visualisation et l'édition de données.

Le système distribué permettra une uniformisation du format. Il acceptera les données dans les différents formats, il permettra également à l'utilisateur expert d'obtenir sa donnée dans le format désiré. Il proposera également à l'utilisateur une interface pour éditer et lire sa donnée.

Exemple : La donnée S6A4_MOUSE (figure 1.5) est de type protéine (et de format fasta). Dans l'exemple, le biologiste devait connaître le format pour repérer les services acceptant directement sa donnée et les services nécessitant la réalisation préalable d'un reformatage.

Désormais, le système gère le passage entre les différents formats. Le biologiste ne doit donc plus savoir que sa donnée est stockée dans le format *fasta* et qu'il faut retirer la première ligne pour obtenir sa donnée en *raw format.*

4.2 Ressources

Une `ressource` est définie comme une information mise à disposition de tous. On pense évidement aux bases de données mais également à certains fichiers aujourd'hui stockés sur les serveurs comme le code génétique [7]. La figure 4.3 représente la modélisation de ce nouveau concept.

3. Unified Model Language est un langage de modélisation défini par l'OMG (Object Management Group).

4. Interface Description Language est un langage de description d'objets distribués neutre [Englebert, 2003].

5. Common Object Request Broker Architecture est une norme définie par l'OMG qui définit l'architecture d'un bus permettant d'échanger des objets de manière transparente [Englebert, 2003].

6. eXtensible Markup Langage permet de créer un document structuré. Ce type de document est utilisé principalement pour le stockage et le transport de données.

7. Le fichier du code génétique contient les correspondances entre les triplets de nucléotides et les acides aminés.

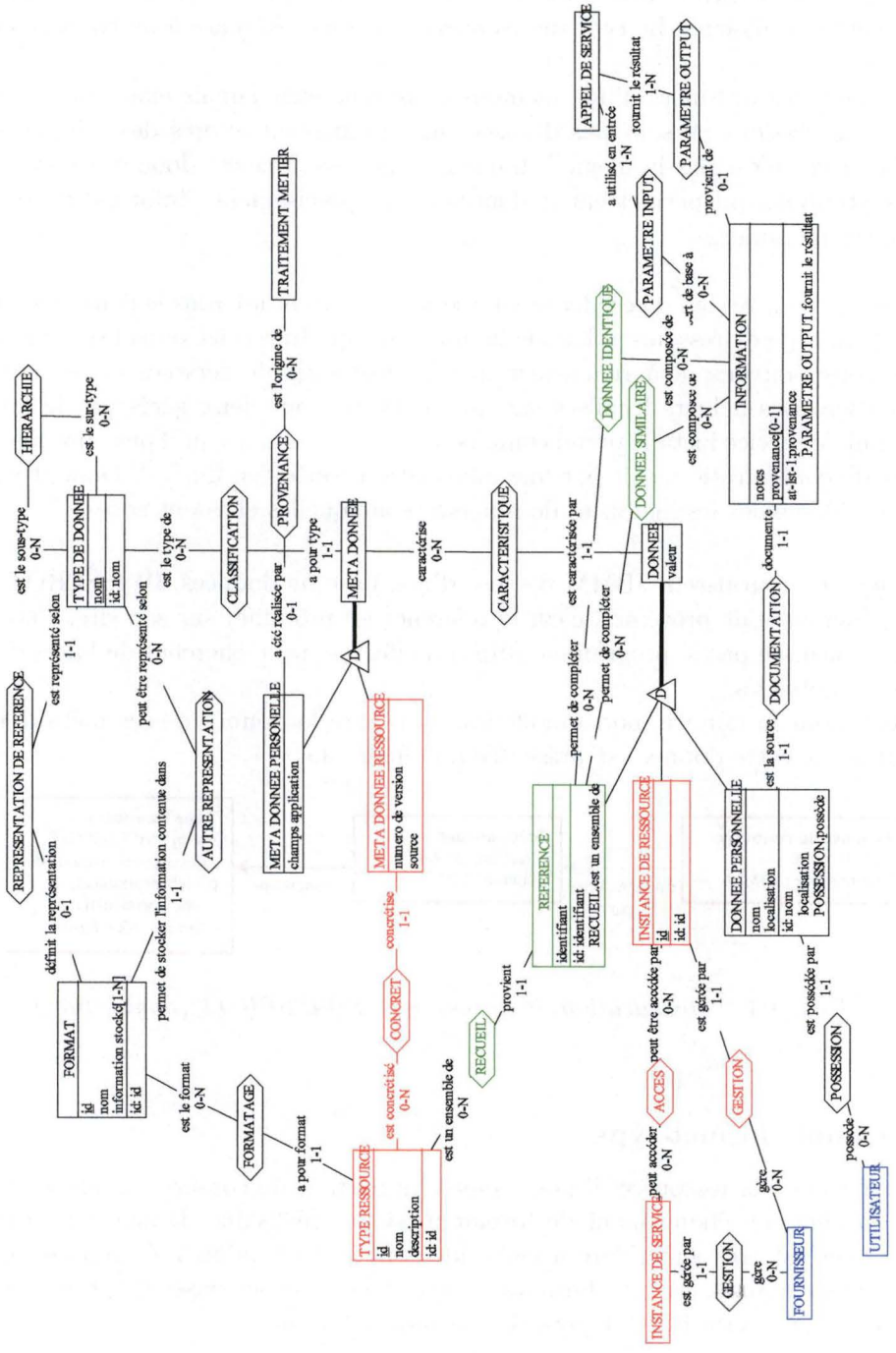


FIG. 4.3 – *Modélisation d'une ressource*

Une instance de ressource est d'abord, tout comme la donnée, de l'information utile aux biologistes. Elle est également caractérisée par un nom, une description, un type et un format. Ces informations ont été regroupées dans le type d'entité TYPE DE RESSOURCE

pour mettre en évidence ce concept d'un même contenu (ou information) disséminé auprès des différents fournisseurs. Le type de ressource fait donc référence à un contenu ou donnée abstraite.

Cependant, les instances d'une même ressource ne sont jamais exactement identiques. Les nouvelles versions ne sont pas diffusées instantanément auprès des différents fournisseurs [8], la source n'est pas la même [9]. L'instance de ressource est donc caractérisée par ces différents attributs qui permettent d'identifier plus précisément l'information présente, la valeur de cette donnée.

Les services ont besoin d'accéder aux ressources. Nous étudierons le lien entre un type de services et un type de ressources lors de la modélisation du service dans le chapitre suivant.

Mais nous pouvons déjà remarquer que les instances de services et les instances de ressources sont des fichiers localisés sur une machine, tout deux gérés par le fournisseur. Aujourd'hui, le service installé prend connaissance des ressources qu'il peut proposer soit par un fichier de configuration, soit par une information codée "en dur" [10]. Donc, les instances de services définissent les instances de ressources auxquelles elles ont accès.

Exemple : Le fournisseur IBMM dispose d'une base de données SWISSPROT (base de données généraliste de protéines, c'est la référence en protéine) sur son site. Cette base de données est utilisée par le programme SRS qui effectue des recherches de bases de données sur base de mots-clés.

Comme pour la donnée, notre modélisation sépare la donnée de ses méta-données. La modélisation de cette donnée est présentée à la figure 4.4.

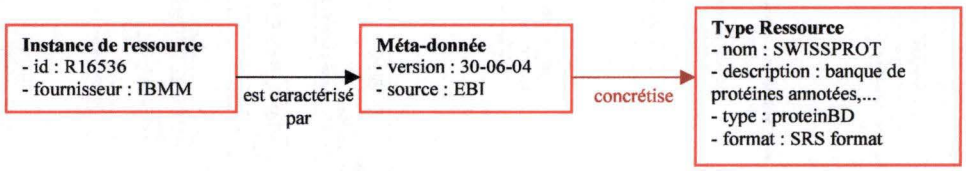


FIG. 4.4 – *Modélisation de la ressource SWISSPROT (application)*

4.2.1 Couple format-type

Dans le cas de la ressource, il nous semble important de conserver l'information sur le format. En effet, un changement de format n'est pas réalisable. D'une part, car la taille des ressources est très supérieure à celle des données [11]et qu'un reformatage serait donc trop long. D'autre part, certains formats ont été créés pour optimiser les performances d'un service. Ainsi, le service BLAST possède son propre format.

Comme la ressource est identifiée par son format, le type de la donnée peut être automatiquement déduit. Cette conclusion simplifie le problème en ne prenant pas en compte la sémantique de la ressource.

8. Le fournisseur décide seul quand il désire télécharger la dernière version de la ressource.
9. Certains fournisseurs ajoutent de l'information qui n'est accessible qu'à leurs membres.
10. La ressource se trouve dans un endroit prédéterminé.
11. La base de données embl contient 40.653.074.943 entrées.

Par exemple, la ressource A est de format $fasta$ et :

- un format $fasta$ stocke l'information d'un type *sequence*
- le type *protein* est un sous-type de *sequence*
- le format $fasta$ ne stocke aucune information sur les attributs spécifiques de *protein*. Seul le code génétique et le nom de la séquence sont stockés.

Nous pourrions tirer comme conclusion que la ressource A est de type *sequence*, or elle est de type *protein*. Le type est donc indispensable pour affiner la sémantique de la ressource.

Cette simplification est courante en bioinformatique car le format est aisé à découvrir alors qu'il est très compliqué de découvrir automatiquement le type précis de la donnée. La vérification des paramètres en entrée est rarement réalisée, le programme essaye simplement de lire la donnée : soit il réussit, soit il échoue et un message d'erreur incompréhensible s'affiche.

4.2.2 Références

Une base de donnée contient une série d'entrées. Chaque entrée constitue une information (avec les annotations). Nous avons voulu prendre en compte cette information sous la forme du concept de **référence**.

Une référence se présente comme embl:P01536 où embl est le nom d'une base de donnée et P01536 est le numéro d'accession ou identifiant. Cette référence correspond à une séquence précise dans un type de resource. Néanmoins d'après la version de la base de données, les annotations seront plus ou moins nombreuses [12].

Précédemment, nous avons précisé que le biologiste tirait de l'information de sa donnée personnelle auprès des bases de données. Cette information provient bien sûr des références. Le biologiste peut trouver :

- les références exactes. Le biologiste trouve une référence correspondant exactement à sa donnée. L'information obtenue dans la référence est donc valable pour la donnée personnelle.
- les références similaires. Le biologiste trouve une référence ayant des points communs avec sa donnée. Une partie de l'information sur la référence est valable pour sa donnée, en faisant certaines hypothèses.

Exemple : La donnée S6A4_MOUSE provient d'une entrée de base de données. Cette donnée sera donc forcément identique à une référence. Cette information est modélisée à la figure 4.5.

La donnée a été trouvée au moyen de SRS, il provient donc d'une ressource avec le format SRS.

12. Une version plus récente peut contenir plus d'informations sur le sens de chaque séquence.

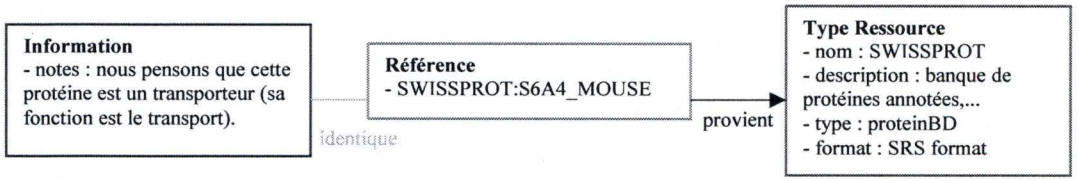


FIG. 4.5 – *Modélisation de la référence S6A4_MOUSE (application)*

4.3 Etude de cas

Pour mieux comprendre les différents concepts, nous allons illustrer notre modélisation en reprenant les différents exemples (voir figure 4.6).

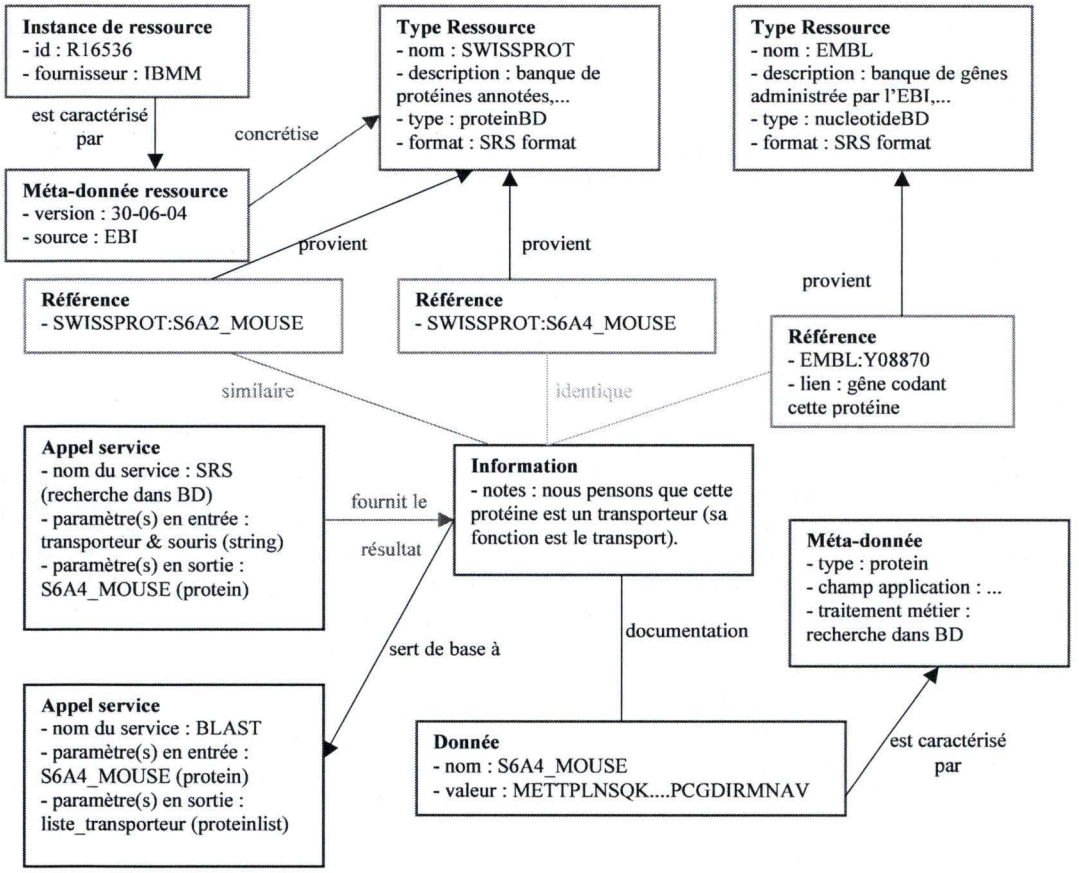


FIG. 4.6 – *Modélisation récapitulative (application)*

Grace aux informations disponibles sur sa donnée dans la base de données SWISSPROT, l'utilisateur connaît le bout d'ADN codant pour sa protéine (cette donnée est également stockée dans une base de données).

L'utilisateur vient de réaliser l'exécution d'un BLAST et il a décidé que la donnée S6A2_MOUSE était similaire à sa donnée.

Ces données sont également présentées dans notre modélisation.

Aujourd'hui, en l'absence d'outil, Bob aurait stocké cette même donnée dans un fichier et le contenu du fichier correspond à la figure 1.5. Seule l'information concernant le nom et la valeur de la séquence aurait été conservée.

4.4 Conclusion

Le biologiste possède des données comme des séquences. L'interprétation de ses données est le but de ces expérience. Nous avons donc réalisé une modélisation permettant de contenir le plus d'informations pertinentes sur la donnée. Cette modélisation devrait aider le biologiste dans son interprétation de ses données. Elle permet de stocker : les appels de service, des notes ainsi que des liens vers les différentes bases de données.

Nous avons également modéliser les ressources comme les bases de données pour mettre en évidence l'information stockée sur le serveur et ainsi accessible à tous.

Ce chapitre a également mis en évidence la difficulté du biologiste de gérer les nombreux formats existants. Heureusement, un système distribué permettrai de gérer ces nombreux formats, donc le biologiste pourrait se concentrer sur ses expériences.

Chapitre 5

Modélisation des services

Le service est un élément essentiel du système. En effet, il correspond à l'offre faite aux utilisateurs. Cette description devra être réalisée par le système avec l'aide de l'auteur et de fournisseurs.

Dans ce chapitre, nous allons étudier les éléments permettant la description d'un service : sa signature, son outil métier (le programme avec lequel il a été réalisé) et son traitement métier (la production/transformation qu'il réalise).

Dans un premier temps, nous étudierons des approches existantes de modélisation de services. Nous établirons ensuite notre catalogue de services. Enfin, nous ajouterons les extensions que l'on peut apporter à ce catalogue en intégrant les informations sur le programme utilisé et sur la transformation effectuée sur les données.

5.1 Modélisations de services existants

Aujourd'hui, les institutions académiques et de recherche offrent aux particuliers l'accès à des services principalement via des pages web. Certains fournisseurs de services modélisent leurs services afin de générer des IHM et/ou exprimer des contraintes d'enchaînement. Par exemple : la suite de programme EMBOSS (European Molecular Bioinformatics Open Software Suite) est constituée d'environ 100 programmes [EBI, 2004c]. Dans le but de gérer l'hétérogénéité, la façon dont l'utilisateur interagit avec le programme (soit ligne de commande, soit interface web ou encore l'aide) est indépendante du programme. Pour générer les différentes interfaces, chaque programme est décrit dans un langage spécialement conçu pour EMBOSS : le langage ACD (Ajax Command Definition).

D'autres fournisseurs ont adopté une autre modélisation comme celle adoptée par le projet Pise [Pasteur, 2004]. Leur modélisation de services a été écrite en XML. Les principaux programmes de bioinformatique ont été représentés dans cette modélisation. Parmi ceux-ci on retrouve les programmes d'EMBOSS, ils ont été transcrits dans la modélisation de Pise grâce à un parseur du fichier ACD; peu d'autres informations ont dû être ajoutées manuellement.On pourrait donc dire que la modélisation d'emboss est au moins aussi riche que le langage ACD.

La modélisation de services devrait également servir à vérifier les types lors de la création d'enchaînement de programmes. En effet, Le type d'une donnée produite est connu mais il

faut que le programme qui utilise cette donnée accepte ce type. Le langage ACD ne permet pas cette vérification mais une ontologie est en phase de développement afin de pallier à ce problème. Par contre, la modélisation utilisée dans le projet Pise permet cette vérification.

5.1.1 Fichier ACD

Cette sous-section constitue le résumé de la référence suivante : [EBI, 2004b]

Chaque programme de la suite EMBOSS est caractérisé par un fichier ACD. Il contient d'abord quelques informations sur le programme :

- une courte description du programme.
- le(s) groupe(s) : constitue(nt) une(des) référence(s) à un classement construit par l'homme qui essaye de grouper les programmes par fonctionnalité.

Ensuite, les paramètres sont décrits. Pour assurer une plus grande lisibilité, les paramètres sont regroupés en sections : input, required, advanced, output. Cette division pourrait être utilisée pour gérer les différents profils d'utilisateurs. Le débutant ne serait intéressé que par les sections input et output, tandis que l'expert désirerait avoir accès à tous ces paramètres.

Pour décrire les paramètres, on dispose des attributs suivants :

- `type`.
  Pour constituer le type du paramètre, il faut se baser sur le genre du paramètre : argument en entrée ou en sortie et l'information exprimée. Ce type est un type de base (entier, etc.), un type biologique, graphique, etc. En se basant sur le type, EMBOSS est capable de lire et d'écrire la donnée dans une série de formats. Cet attribut est le seul obligatoire.
- `information` : définit le nom du paramètre présenté à l'utilisateur.
- `default` : donne la valeur par défaut.
- `description` : explique le rôle du paramètre.
- `missing` : indique si l'attribut peut prendre la valeur null.
- `standard` : indiques si la valeur est obligatoire ou non.

ainsi que d'autres attributs n'ayant pas d'intérêt dans cette analyse.

Il existe également d'autres attributs propres à chaque type. Ils permettent de rajouter des contraintes sur le type. Par exemple pour le type entier, on peut spécifier un minimum et/ou un maximum.

Le langage ACD n'est pas qu'une simple énumération des propriétés,il permet également d'exprimer des expressions logiques. En effet, des opérations basiques ont été intégrées dans ce langage tel que les opérations arithmétiques et les opérations conditionnelles (opérations sur les booléens, *if then else* et le *case*). Ces opérations sont effectuées sur des constantes et sur des attributs calculés à partir des différents paramètres du service. On peut ainsi définir la valeur par défaut d'un paramètre b comme étant la moitié de la longueur de la séquence a. Ces expressions sont très utiles pour exprimer les valeurs par défaut, les valeurs de contraintes ainsi que pour exprimer si un paramètre doit être présenté à l'utilisateur.

Chaque type possède également des attributs qui peuvent être fixés dans la ligne de commande. Ils permettent à l'utilisateur de paramétrer chacun des arguments du service. Ainsi, on peut définir pour une séquence (chaîne de caractères) son début et sa fin, le programme ne prendra en compte qu'une partie de la séquence (sous-chaîne de caractères). Actuellement les IHM prennent en compte ces attributs en fournissant pour chaque argument plusieurs champs de saisie.

Pour conclure, nous formulerons quelques critiques. On constate que ce langage est fort riche : énormément d'informations ont pu être exprimées mais certaines informations ne sont pas utilisées. Par exemple, peu d'interfaces graphiques savent calculer les expressions logiques, elles ne sont donc pas valuées.

D'autres informations sont mal exprimées, donc non utilisables, car la sémantique exacte des attributs n'est pas définie. Ainsi, aucune règle n'a été établie pour ranger un paramètre dans une section plutôt que dans une autre. L'appréciation dépend de la personne qui rédige.

Enfin, cette modélisation a été principalement axée sur les utilisateurs expérimentés. La masse d'options (comme les attributs que l'on fixe à la ligne de commande), très utile à l'expert, génère des interfaces qui découragent les utilisateurs débutants et normaux. Il faut donc prévoir pour un même service différents profils qui mettront ainsi les interfaces de services à la portée de tous les types d'utilisateurs.

5.2 Modélisation de notre catalogue de service

Cette section est basée sur la première architecture réalisée dans le cadre de [Buyle et Dallons, 2003]

Nous souhaitons intégrer dans notre catalogue non seulement les types de services offerts à l'utilisateur mais également les différents emplacements proposant ces services et enfin intégrer les informations enregistrées lors d'un appel à un service. De telle sorte que notre catalogue serve à la recherche de services (recherche par l'utilisateur, mais aussi par le système pour trouver la meilleure localisation pour un service donné), à la génération d'IHM, à la construction d'enchaînement de services et à la ré-exécution d'appel.

Nous allons analyser les différents éléments constitutifs du catalogue :

- le **service** : autrement dit une application, celle-ci est caractérisée par ses paramètres, etc.;
- le lien avec les **ressources** : comment un service connaît les ressources qui pourront lui être utiles;
- l'**instance de service installée** : correspond à une application sur une machine précise, celle-ci est caractérisée par sa localisation, etc.;
- l'**instance de service invoquée** : correspond à un appel réalisé par un utilisateur. Les caractéristiques d'un appel sont les valeurs des paramètres, les résultats obtenus, l'état de terminaison, etc.

Après avoir constitué un catalogue classique, nous allons examiner les spécificités de la bioinformatique en terme d'outils métier, de traitements métier et les implications dans la

modélisation de notre catalogue.

Notre approche itérative sera complétée par des schémas entité-association qui évolueront avec les éléments ajoutés.

Pour simplifier ces différents schémas, seule l'information pertinente est représentée. Les éléments ajoutés au schéma sont représentés en rouge. Enfin, un schéma récapitulatif se trouve à la fin du chapitre (page 68).

5.2.1 Service

La figure 5.2 (page 49) représente notre modélisation du service.

D'abord, nous allons supposer qu'un service correspond à un programme comme dans la modélisation ACD. Nous pouvons donc représenter ce programme par une série de caractéristiques : son nom, sa fonction, etc.

Cette fonction se définit par une signature qui est l'ensemble des méta-données de chacun de ses paramètres tant en entrée qu'en sortie.

En biologie, les programmes transforment des données ou produisent de nouvelles données. L'utilisateur a accès à des paramètres lui permettant d'influencer cette transformation ou production mais l'information principale reste le flux de données. On peut représenter cela par la figure 5.1.

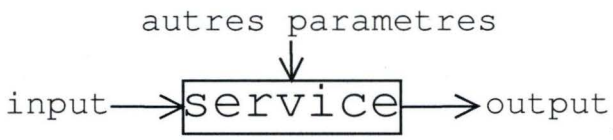


FIG. 5.1 – *Représentation du service par les biologistes*

La signature du service sera donc définie comme :

f(input, autres paramètres[1]) = output

La figure 5.2 propose une modélisation de cette information en répartissant les paramètres en trois catégories. Une signature demande des données pour en produire de nouvelles en utilisant d'autres données pour influencer le comportement de l'algorithme. Les données influençant le processus sont, en général, des données moins significatives biologiquement.

Dans le chapitre 3, nous avions introduit deux vues du service : une générale reprenant les caractéristiques et une utilisateur décrivant ses attentes.

Cette distinction entre la vue générale du service et la vue de l'utilisateur se fait au niveau de la signature. Un service peut posséder plusieurs signatures qui décrivent chacune un niveau de connaissance : débutant, normal ou expert. La signature de niveau normal est

1. L'ensemble *autres paramètres* reprend les éléments des sections : normal, advanced dans la modélisation ACD.

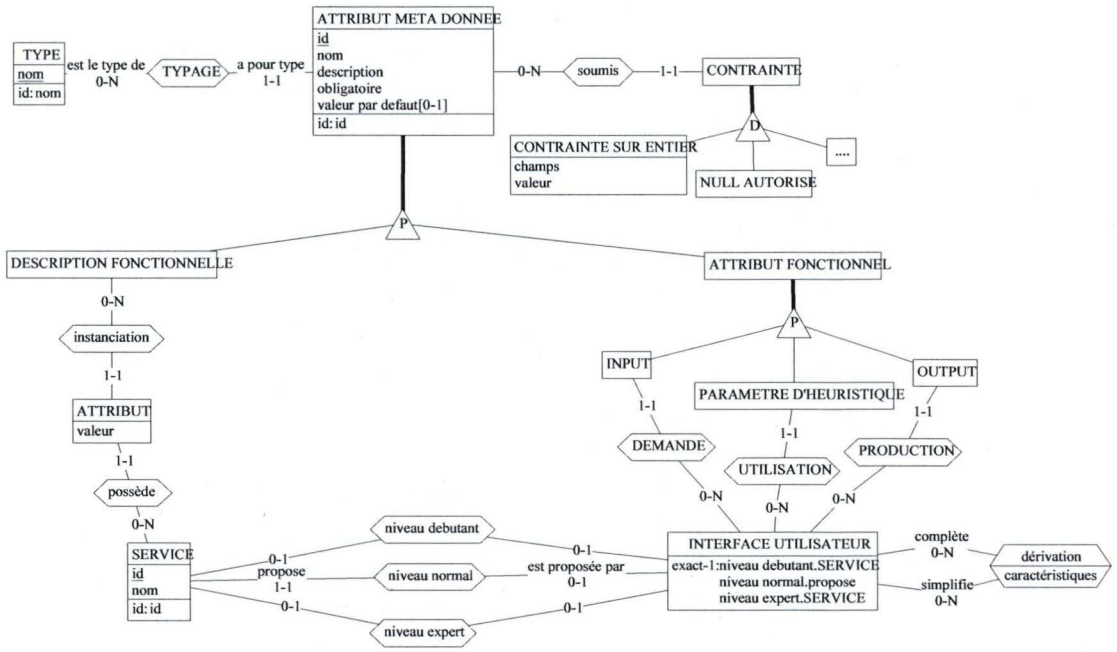


FIG. 5.2 – *Modélisation de services*

obligatoire de sorte que tout service possède au minimum une signature.

Ces signatures doivent découler l'une de l'autre autrement dit :

- service ∘ niveau débutant ∘ simplifie = service ∘ niveau normal
- service ∘ niveau expert ∘ complète = service ∘ niveau normal

Chaque signature est donc une simplification ou une généralisation d'une autre signature du service. Cette simplification ou généralisation de signature doit être documentée par des caractéristiques. Ces caractéristiques permettent, à partir d'une signature, de retrouver l'autre.

Supposons un service X, celui-ci pourrait être décrit (par exemple) par ces signatures :

1. une signature de niveau normal :

$$f_n\ (\alpha\ ,\ \delta\ ,\ \epsilon\ ,\ \zeta\ [0-1]) = \gamma\ ,\ \iota$$

2. une signature de niveau débutant :

$$f_d\ (\alpha\ ,\ \beta) = \gamma$$

- **caractéristiques de dérivation :**

" if ($\beta == 0$) then $\delta = 2$, $\epsilon = 4$
else $\delta = 1$, $\epsilon = 6$ "

La signature 2 simplifie la signature 1, la rendant plus accessible pour les débutants. Cette signature simplifiée est toujours exprimable dans la signature complète par la transformation stockée dans les caractéristiques de dérivation.

L'entité **Attribut Meta Data** exprime les méta-données des paramètres de la fonction. Ces méta-données sont des données sur les données (ou paramètres), elles sont composées des informations suivantes :

- `nom` : nom du paramètre.
- `description` : courte explication du paramètre.
- `obligatoire` : paramètre obligatoire ou non.
  Si un attribut descriptif est obligatoire alors chaque service doit posséder cet attribut.
- `valeur par défaut` : valeur utilisée si l'utilisateur ne spécifie pas de valeur.
- `type` : type de donnée.
- `contraintes` : expression logique permettant de limiter le domaine des paramètres (dans le cas des input, les contraintes sont les pré-conditions).
  Ces contraintes peuvent exprimer un lien avec les types de données, ainsi les éléments de CONTRAINTE SUR ENTIER représentent les contraintes sur le domaine entier.

Ces champs doivent pouvoir être remplis sous la forme d'un petit langage basique (similaire à celui de l'ACD). En effet, les informations exprimables avec ce langage sont intéressantes : elle permettent de fixer dynamiquement les caractéristiques [2] d'un paramètre en fonction des autres paramètres du service.

Pour profiter efficacement de ces expressions, une mise à jour des contraintes doit être effectuée lors des appels. Supposons les contraintes suivantes définies pour le service X :

- $type_\alpha = any$ *(protein ou nucleotide)*
- $type_\delta = \text{if}(type_\alpha = protein)$ then *protein* else *nucleotide*

tant que le paramètre α n'a pas été évalué, les valeurs acceptées pour δ sont de type *any*. Dès que l'utilisateur introduit une valeur pour α de type *protein*, les valeurs acceptées pour δ seront de type *protein*, et ce tant que le type de α reste *protein*.

Enfin, un service est décrit par des attributs qui sont eux-mêmes décrits par l'entité `Description Fonctionnels`. La table 5.1 reprend quelques éléments descriptifs d'un service.

nom	**description**	**type**	**obligatoire**	**valeur par défaut**	**contraintes**
nom	nom du service	string	oui		
description	description du service	string			
traitement métier	production/transformation de données				
algorithme	algorithme utilisé	algorithme			
précision	pourcentage indiquant si la prédiction est juste	entier			
champs d'application	domaine où le service est utile				

TAB. 5.1 – *Description fonctionnelle d'un service*

2. Les caractéristiques sont : la valeur par défaut, le type, la présence d'un paramètre, les bornes du domaine, etc.

5.2.2 Lien ressource-service

L'utilisateur doit, en général, utiliser un service pour accéder aux ressources comme par exemple un service de recherche,... Les services ont également besoin d'accéder à des informations stockées sur le serveur[3].

Un service désirant accéder à une ressource doit définir le type d'informations dont il a besoin ainsi que les formats qu'il sait lire. Ces informations permettront de définir l'ensemble des types de ressources acceptables (voir figure 5.3).

Définir les ressources acceptables sur base du type et du format est une solution plus complète que de définir les ressources acceptables directement, autrement dit avoir une association many-to-many entre service et type de ressources.

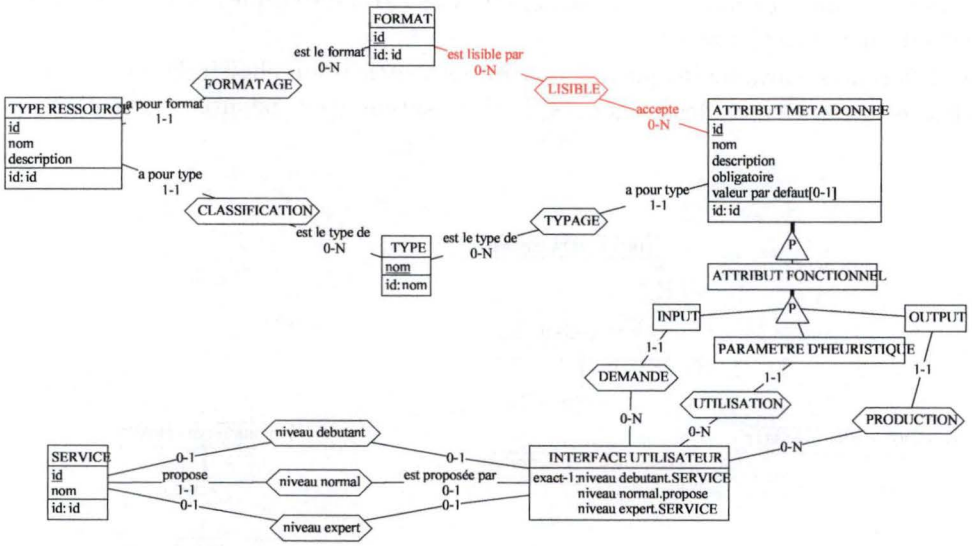


FIG. 5.3 – *Lien du service avec les ressources*

Supposons un service de recherche : X, celui-ci a un paramètre α de type Y qui accepte le format Z. Aujourd'hui, pour le paramètre α, l'utilisateur a le choix entre une série de *bases de données ou ressources* disponibles pour ce service pour effectuer sa tâche. Le fournisseur réalise donc un lien direct entre le service et les ressources. Nous pouvons donc définir :

$A = \{$ensemble des ressources de type Y encodées dans le format $Z\}$

$B = \{$ensemble des ressources proposées pour α dans le service $X\}$

- avec $B \subseteq A$

  sinon l'ensemble A contient des éléments qui ne sont ni de type Y, ni de format Z et donc la déclaration du paramètre α est erronée et doit ˆ

La liste des ressources n'est donc plus dépendante d'un service. Ce qui va permettre une gestion indépendante et plus aisée des ressources par l'administrateur et va, de plus, permettre à l'utilisateur d'avoir une vue claire de ce qu'il pourrait réaliser.

5.2.3 Instance de service

La figure 5.4 (page 52) représente le catalogue de service intégrant le concept d'instance de service.

Nous introduisons ici le concept de localisation. En effet, un service peut être disponible sur différentes machines. Chacune de ces machines ayant des caractéristiques différentes : un propriétaire, une certaine performance, etc. Ces caractéristiques seront représentées par des attributs non fonctionnels[4].

Ces différentes caractéristiques seront importantes pour choisir la meilleure instance étant donné un service et des exigences de l'utilisateur (par exemple : gratuit et assez rapide).

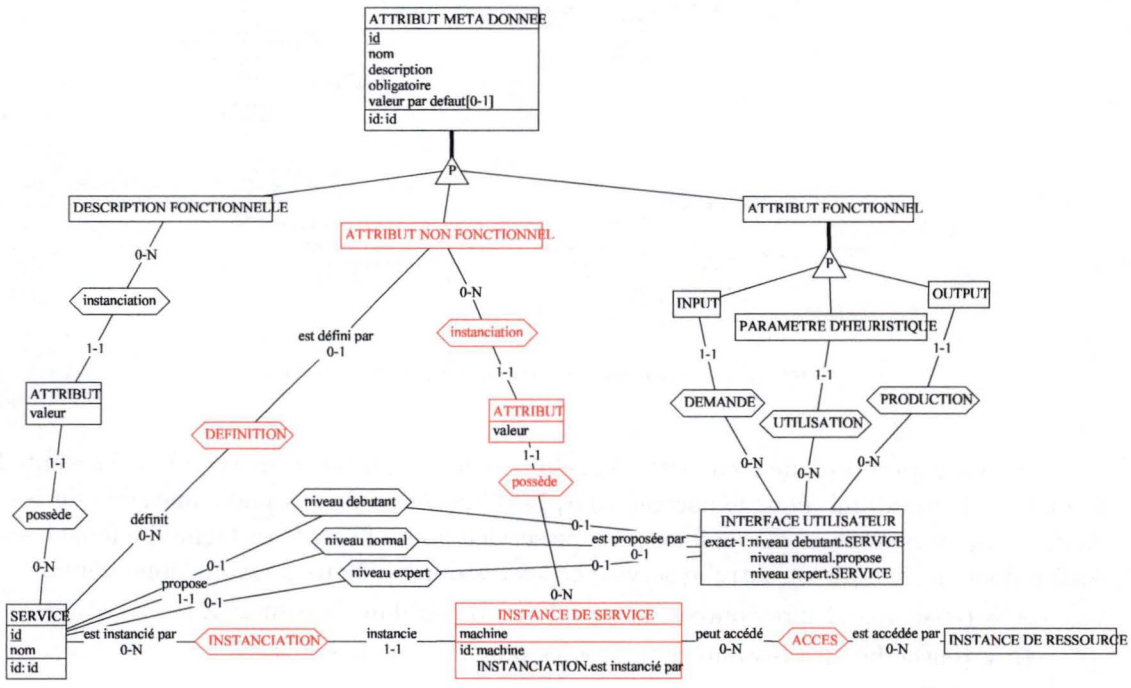


FIG. 5.4 – *Modélisation d'instances de services*

Le service définit les attributs fonctionnels ainsi que les valeurs que ces attributs peuvent prendre, l'instance définit les différentes valeurs de ces attributs.

4. Ces attributs sont appelés non fonctionnels car leurs impacts ne sont pas directs sur la fonctionnalité offerte à l'utilisateur.

Cela permet, par exemple, au service de définir sa consommation CPU [5] mais la valeur de cet attribut peut être modifiée d'après les capacités de la machine sur laquelle le service est installé.

L'attribut dynamique a été ajouté pour tenir compte des attributs qui changent souvent comme l'attribut *occupé*. En jugeant un attribut dynamique, l'administrateur délègue au système le devoir de mettre à jour l'attribut lors de chaque évolution.

La table 5.2 reprend les principaux attributs non fonctionnels des instances de service.

nom	**description**	**type**	**obligatoire**	**valeur par défaut**	**contraintes**	**dynamique**
propriétaire	fournisseur d'accès					
installation	type de machine					
performance	mesure des performances du service sur la machine					
occupé	disponibilité immédiate pour exécuter un appel					oui
version	version du service					

TAB. 5.2 – *Attribut non fonctionnel d'une instance de service*

Remarque : pourquoi considère-t-on la version comme un paramètre de l'instance et non du service ? Car un changement de version peut avoir deux conséquences : soit la fonctionnalité a changé, soit des légères modifications ont été apportées au code. Dans le premier cas, un nouveau service sera créé. Dans le second cas, on considère que le service offert reste le même.

Nous avons déjà défini qu'une instance de service était capable communiquer avec une instance de ressource si celle-ci se trouvaient sur la même machine (par un lien many-to-many entre `instance de service` et `instance de ressource`).

Une instance de service dispose donc d'un ensemble de ressources qui possède la propriété suivante :

$A = \{$ ressource de type Y et de format $Z\}$

$B = \{$instance ressource de type Y et de format Z accessible depuis l'instance de service $X\}$

$B \subseteq A$, avec X, Y et Z quelconques

Tout naturellement, le choix du service dépendra des attributs non fonctionnels et des **instances de ressources** disponibles sur la machine. En effet, l'utilisateur souhaite réaliser le service X avec un type de ressource Y comme valeur à un paramètre. Seules les instances de service X possédant une instance de la ressource Y pourront être choisies.

5. Pour définir sa consommation CPU, le fournisseur crée un attribut non fonctionnel avec comme valeur par défaut la valeur désirée.

5.2.4 Appel de service

La figure 5.5 (page 54) représente le catalogue comprenant également les informations sur les appels des différents utilisateurs.

L'utilisateur est enfin capable de faire un appel à un service. Ces informations vont permettre de réaliser des audits du système, d'enregistrer les conditions d'expérience; celles-ci vont permettre de répéter l'expérience de telle façon qu'elles fournissent les mêmes résultats[6]. Nous avons besoin de connaître :

- les valeurs des paramètres d'input
- les valeurs des paramètres d'output
- toutes les informations sur le programme utilisé : correspond aux informations sur SERVICE et INSTANCE SERVICE.
- l'état de l'appel (EN COURS, FINI, etc.)
- les erreurs survenues
- la date de l'appel
- l'utilisateur initiateur de l'appel.

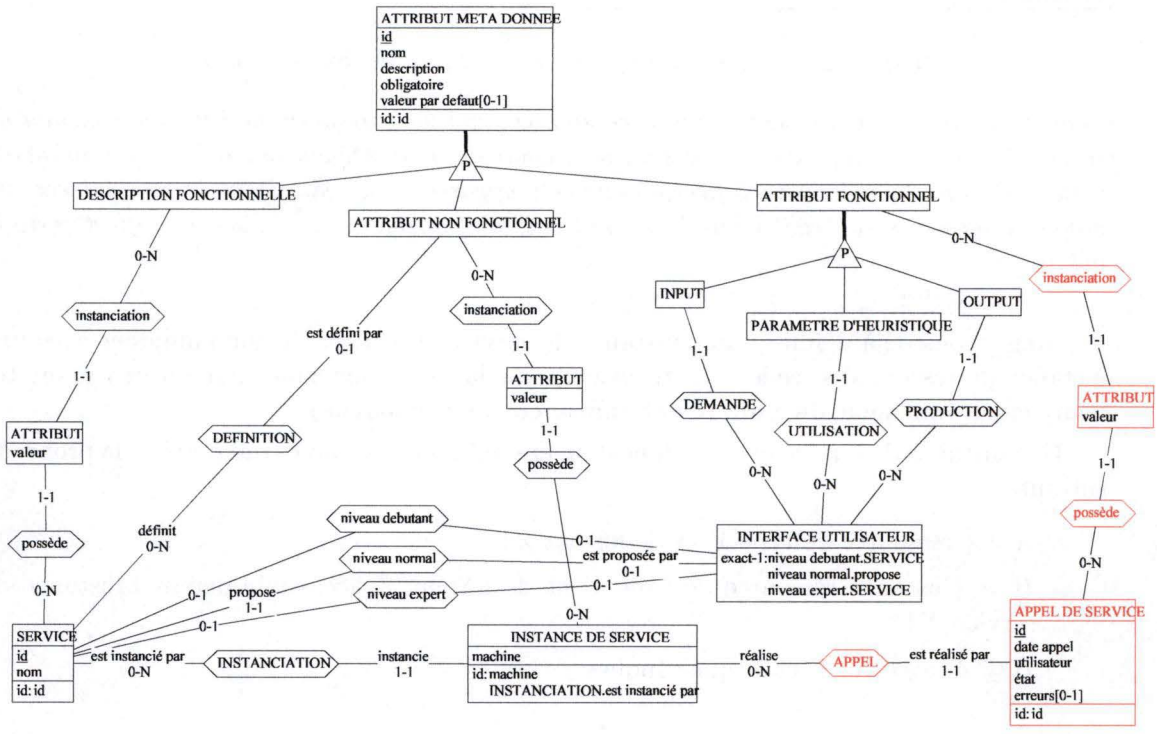


FIG. 5.5 – *Modélisation d'un appel de service*

6. Le stockage des informations sur une exécution permet de créer un équivalent fonctionnel au cahier de laboratoire. Durant chacune de ses manipulations dans son laboratoire, le biologiste écrit précisément les différentes étapes ainsi qu'éventuellement les éléments qui ont fait rater ou réussir son expérience.

5.2.5 Etude de cas

La figure 5.6 reprend la modélisation du service BLAST. Ce service propose à l'utilisateur deux signatures : une de niveau débutant, l'autre de niveau normal.

Deux fournisseurs proposent ce service : l'IBMM et l'EBI. Les fournisseurs garantissent la même performance que le service, cet attribut n'est donc pas redéfini par les instances. La version fait partie de la fourchette fixée par le service. Seule l'instance de service I156432 a accès à une instance de ressource, elle est donc la seule capable aujourd'hui de réaliser un appel avec comme paramètre : la resource SWISSPROT.

L'utilisateur Bob a d'ailleurs réalisé un appel sur cette instance de service. Cet appel est maintenant terminé et s'est produit sans erreurs. On retient les différents paramètres remplis par l'utilisateur pour lui permettre de ré-exécuter son appel dans les mêmes conditions (avec le même service et le même type d'instance).

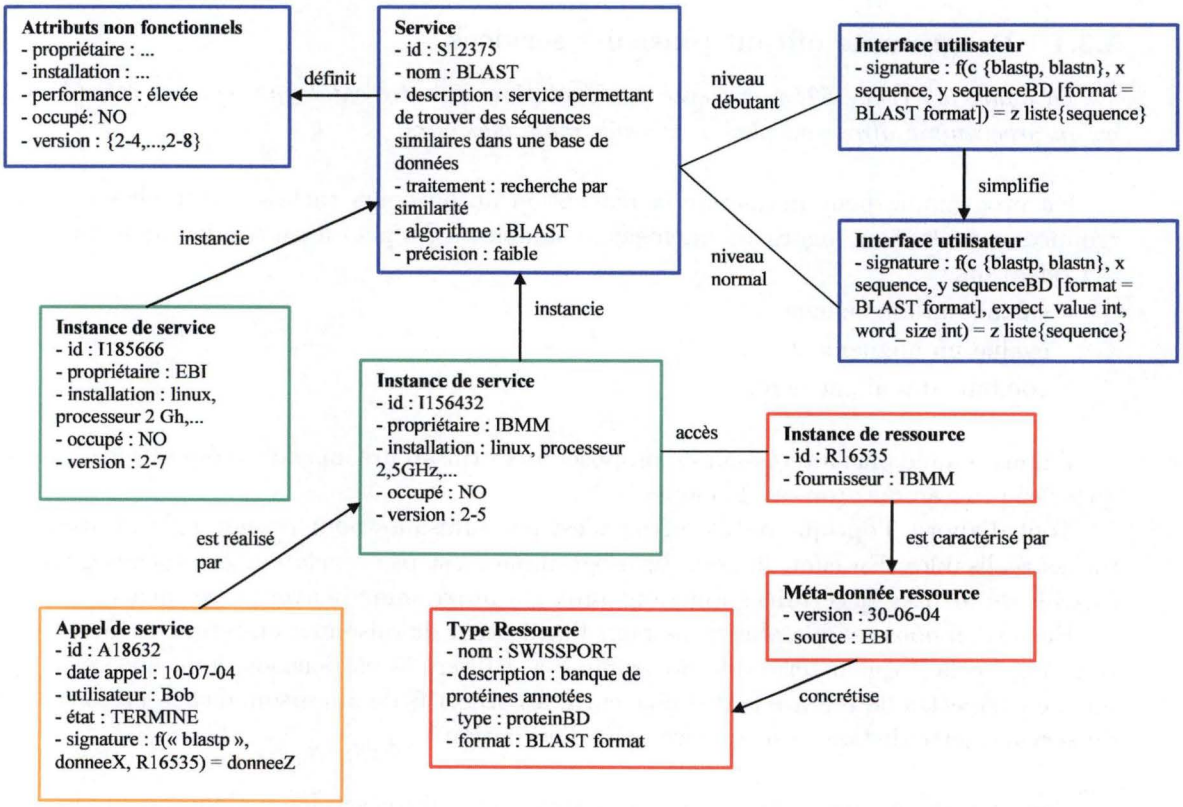


FIG. 5.6 – *Modélisation du service BLAST, d'instances et d'un appel (application)*

5.3 Outils métier

Aujourd'hui, l'approche employée par les fournisseurs de services est généralement une approche orientée outils métier ou programme, autrement dit une interface correspond à un programme.

De plus, ces programmes sont souvent regroupés en package et les interfaces correspondant à ces programmes sont similaires.

Ces informations sont donc essentielles dans l'utilisation actuelle de la bioinformatique. Ces éléments doivent donc être intégrés dans notre catalogue de service. De plus, il nous semble essentiel de relier les éléments à leur fonction et de ne plus considérer un programme comme un service. Le programme permet de réaliser le service. Le service est ce qu'on offre à l'utilisateur.

Nous allons d'abord étudier la différence "programme-service". Nous étudierons ensuite l'information apportée par le concept de package.

5.3.1 Programme offrant plusieurs services

La figure 5.7 (page 57) représente une évolution de notre catalogue, si nous considérons qu'un programme offre non plus un service mais plusieurs.

Un programme peut permettre la réalisation de plusieurs tâches. Ces tâches sont regroupées car elles sont algorithmiquement semblables [7]. On peut ainsi trouver un algorithme (TCoffee) qui :

- calcule un alignement
- évalue un alignement
- combine des alignements

Il nous semble plus intéressant de proposer aux utilisateurs un accès à des services plutôt qu'à des programmes (ou outils métier).

Tout d'abord, l'optique outils métier n'est pas suffisante pour révéler à l'utilisateur les tâches réalisables. En effet, le nom du programme n'est pas révélateur des tâches qu'il est capable de réaliser et certaines fonctionnalités du programme peuvent être cachées.

Ensuite, il nous semble plus facile pour l'utilisateur de raisonner en terme de tâches ("je veux faire cela") qu'en terme de programme à utiliser. Nous pensons donc que cette approche permettra de réduire la distance entre les objectifs de la personne et la présentation du service, cette distance est appelée golfe d'exécution [8].

Pour mettre en place un accès à des services, l'entité service a donc été divisée en deux : d'une part, l'implémentation, elle correspond à la représentation du programme (ex : TCoffee) et d'autre part, le service qui correspond à ce qu'on offre à l'utilisateur (ex: calculer un alignement multiple, évaluer un alignement, combiner des alignements,...)

De même l'entité instance de service a été divisée. Ces nouvelles entités sont représentées dans la figure 5.7. Les éléments en bleu et en vert sont respectivement les caractéristiques

7. Ces tâches ont une majorité de lignes de code en commun.

8. Un golfe d'exécution correspond à la distance entre les objectifs de la personne et les variables d'action du système physique, autrement dit entre la tâche que l'utilisateur veut accomplir et ce que le système lui propose [Bodard, 2002].

liées à l'implémentation et au service.

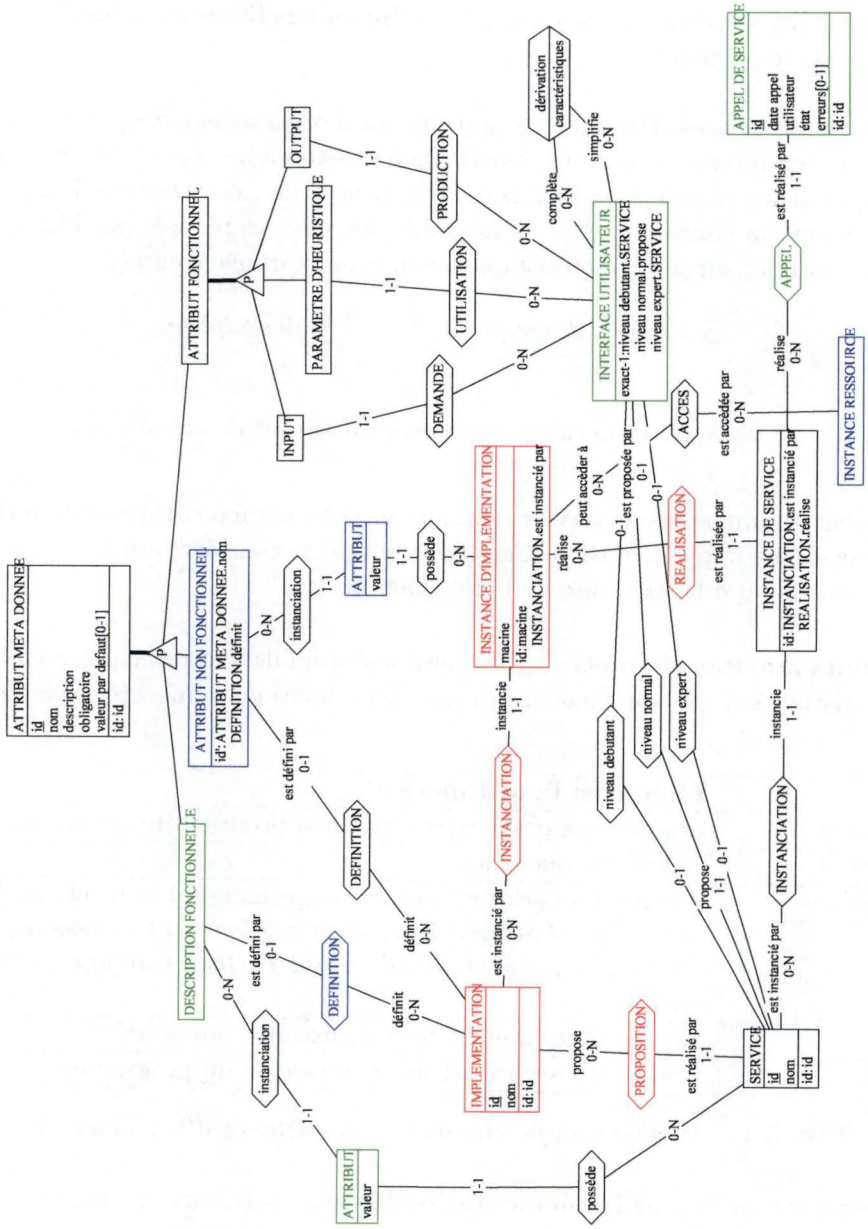


FIG. 5.7 – *Modélisation des programmes donnant accès à plusieurs services*

Le lien entre service et implémentation. L'entité implémentation définit les attributs descriptifs de l'entité SERVICE ainsi que les valeurs que cet attribut peut prendre. Le service définit éventuellement la valeur de ces attributs.

Cela permet à l'implémentation de définir le programme utilisé[9] afin que le service ne doive plus redéfinir cette valeur.

La définition du service se base sur la description fonctionnelle et les différentes interfaces utilisateur (ou signatures).

Les différents services d'une même implémentation possèdent désormais leurs propres signatures et leur propre description fonctionnelle. Néanmoins, ces différents services proviennent de la même implémentation, ils doivent donc avoir des caractéristiques communes.

Soit la définition suivante : $S(i)$ = l'ensemble des services réalisés par l'implémentation i, alors la contrainte sur les caractéristiques peut être exprimée comme:

$$\exists\, s \in S \;\dagger\; description_s{}^{10} = \bigcup_{x \in S} description_x$$

$$\cap\, \forall\, x \in S,\; signature_s{}^{11} \textit{ est généralisation de } signature_x$$

Autrement dit, au moins un service reprend toutes les caractéristiques des autres services offerts par la même implémentation. Ce service peut être considéré comme la "somme" des autres, il correspond à la signature de l'implémentation.

Les attributs non fonctionnels : nous considérons qu'ils sont principalement dépendants des caractéristiques de l'algorithme et des caractéristiques de la machine(voir tableau 5.3).

Nom	Lien avec l'outil métier
propriétaire	propriétaire des services est aussi propriétaire du programme
installation	type de la machine
performance	mesure des performances du programme sur la machine. *Nous faisons l'hypothèse que le code est relativement homogène, donc un calcul d'alignement prend autant de temps qu'une validation de résultat*
occupé	disponibilité immédiate pour exécuter un appel[12]
version	version du service est aussi la version du programme

TAB. 5.3 – *Attribut non fonctionnel d'une instance d'implémentation*

Pour cette raison, l'entité **implémentation** fixera les attributs non fonctionnels et l'instance d'implémentation les instanciera.

Il ne sera donc pas possible de trouver deux instances de services provenant de la même implémentation avec, par exemple, deux numéros de version différents. Le catalogue ne pourra donc pas être incohérent.

9. L'implémentation fixe un attribut qui prend comme valeur par défaut son nom et qui n'accepte que ce nom comme valeur.

10. $sdescription_s$ correspond à la description du service s.

11. $signature_s$ correspond à la signature du service s.

12. Cette disponibilité dépend aussi des autres programmes installés sur la machine.

Afin de mener une politique de différenciation (en terme de prix, etc.) entre les attributs non fonctionnels de différents services ayant une même implémentation, le fournisseur peut définir la valeur par défaut, les valeurs possibles de l'attribut à l'aide du langage de description.

Ainsi un fournisseur, désirant interdire l'accès au service X, fournira comme contrainte à l'attribut occupé : $if(service == X)\ then\ YES$ [13].

5.3.2 Notion de package.

Aujourd'hui, certains auteurs regroupent leurs services en package (ou suite de programmes). Cette information est devenue importante pour l'utilisateur car ces programmes sont regroupés sur un même point d'accès (une même adresse web).

Comme deux mêmes instances d'un même package contiennent exactement les mêmes implémentations, il pourrait être intéressant de rajouter une entité package (voir figure 5.8) afin de faciliter le travail d'encodage lors de l'installation d'un certain package dans la fédération.

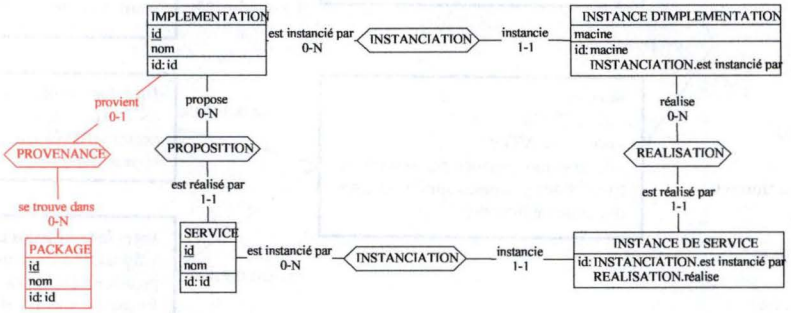


FIG. 5.8 – *Modélisation d'un package*

Cette solution n'apporte que peu d'informations surtout si l'on considère que l'information ajoutée n'est utile fondamentalement que lors d'une installation. En effet, le package ne donne aucune indication sur la description d'un service ou sur les attributs non fonctionnels d'une instance de service.

Néanmoins, l'utilisateur a besoin de cette information mais au même titre que la tâche du programme (description du service). Nous pouvons donc introduire le nom de package comme un attribut descriptif du service.

5.3.3 Etude de cas

BLAST est un programme permettant de rechercher dans une base de données toutes les séquences similaires à une séquence requête. Le premier paramètre demande à l'utilisateur de choisir parmi des " sous-`service` ": `blastp` permet, à partir d'une protéine, de trouver des protéines similaires dans une base de données et `blastn` permet la même opération mais

13. Si le service demandé est X, alors répondre qu'il est occupé.

pour les nucléotides.

Nous allons donc définir dans notre catalogue de `services` (voir figure 5.9) : l'implémentation BLAST qui `proposes` deux services[14] : `blast` (prend en entrée protéines et nucléotides) et `blastp` (prend uniquement en entrée des protéines).

On constate d'abord une simplification dans les signatures. En effet, le premier paramètre : *c* n'apparaît plus dans le service `blastp` : il a été fixé à la valeur `blastp`.

Ensuite, l'implémentation a déjà défini les attributs de description : `traitement`, `algorithme` et `précision`. De sorte que les services ne doivent plus les définir.

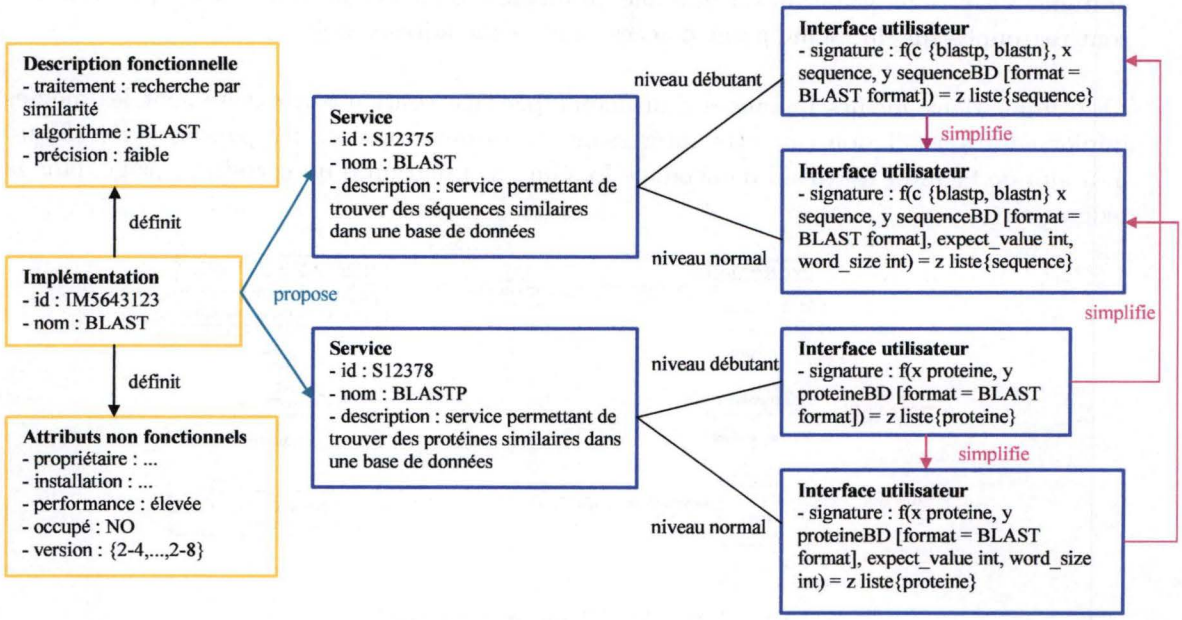


FIG. 5.9 – *Modélisation de l'implémentation BLAST proposant deux services : BLAST et BLASTP (application)*

5.4 Traitement métier

Le traitement métier exprime la transformation (ou production) de données. Cette transformation demande et produit des données. Chaque service réalise un traitement. Par exemple, `BLAST` réalise une recherche de similarité, c'est une production de séquences similaires aux séquences données en entrée. De plus, le traitement établit un lien entre les données (en entrée et en sortie). Dans le cas de `BLAST`, le lien est : "similaire à".

Le traitement exprime l'action du service. L'utilisateur utilise donc cette information lors de ses recherches de service. Le traitement est d'ailleurs le premier critère de choix d'un service.

14. En fait, il définit trois services : blast, blastp et blastn mais pour ne pas surcharger le schéma, nous avons omis le troisième.

Aujourd'hui, l'information sur le traitement est surtout localisée dans les classifications répartissant les services. Or cette information est fondamentale. En effet, savoir utiliser la bioinformatique correspond d'abord à la maîtrise des traitements.

Nous aimerions mettre l'accent sur le traitement dans notre catalogue d'abord par l'ajout d'une classification. Ensuite, nous étudierons la possibilité de créer des services représentant les différents traitements.

5.4.1 Classification

Dans l'analyse de la situation actuelle (voir section 3.1), nous avions mis en évidence la classification qui présentait les services d'un fournisseur. Chaque fournisseur propose sa classification. Par exemple, la figure 5.10 représente une classification des services `blast`, `blastp` et `blastn`.

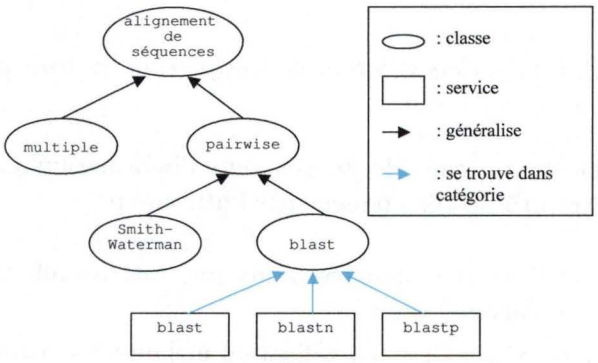


FIG. 5.10 – *Classification existante des services : blast, blastn et blastp [Wroe et al., 2003]*

Or cette classification ne permet pas d'identifier certaines informations sur le service ainsi `blastp` demande uniquement des protéines et `blastn` uniquement des nucléotides. D'autres classifications prennent en compte le type en entrée mais omettent sûrement d'autres informations.

En bioinformatique, on trouve beaucoup de classifications différentes car on peut caractériser le service par trois axes :

- traitement : correspond à la création/transformation de données.
- algorithme : décrit le processus de réalisation du traitement. L'algorithme décrit les étapes, on l'apprend au biologiste pour qu'il comprenne le cheminent de sa donnée et qu'il soit capable de réaliser un petit exemple sur papier.
- donnée : spécifie les données attendues en entrée.

Chaque fournisseur pioche dans ces différents axes pour réaliser sa classification. Pour construire la figure 5.10, le fournisseur a d'abord utilisé l'axe traitement puis l'axe algorithme.

Pour constituer toute l'information utilisée par les fournisseurs pour créer la classification, il suffit de caractériser chaque service par les trois axes.

Prenons un exemple, le service **blastp** est un service réalisant un alignement pairwise [15] local, il prend en entrée deux protéines. Le service utilise un algorithme de type blast, cet algorithme est de type heuristique autrement dit le programme répond rapidement mais le résultat est légèrement imprécis. La triple classification de ce service a été réalisée à la figure 5.11.

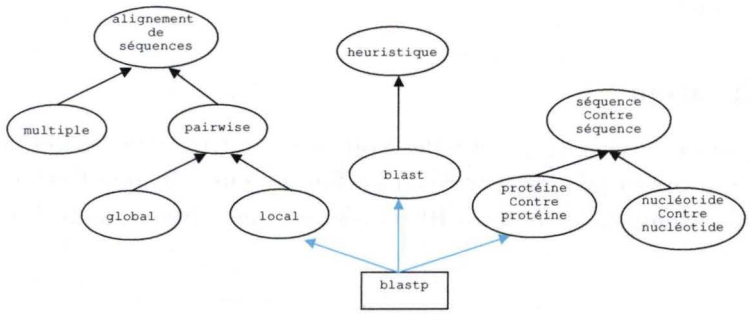


FIG. 5.11 – *Classification de blastp selon les trois axes*

Cette triple classification permet de stocker toute l'information mais fait intervenir trop de dimensions pour être utilisée directement par l'utilisateur.

Les classifications utilisées par les fournisseurs sont intéressantes. Il serait opportun de pouvoir réutiliser leurs caractéristiques :

- classification multi-axiale. Cette classification mélange les classes des différents axes : traitement, programme, donnée comme la figure 5.10.
- les multiples possibilités. Nous voudrions pouvoir classifier notre catalogue sur base de plusieurs classifications pour que la classification s'adapte à la situation de l'utilisateur. En effet, il est ridicule de proposer à un biologiste ayant choisi de travailler sur une protéine une catégorie de services ne s'appliquant qu'aux nucléotides.

Pour mettre en place ces caractéristiques, nous avons défini le concept de type de classification. Chaque classification est définie par un type. Ce type définit l'enchaînement des axes de classification ainsi que le niveau de détail de ces différentes catégories [16].

Une autre caractéristique des classifications de services utilisées aujourd'hui est leur réalisation a priori. La gestion de ces classifications est en général faite manuellement lors de l'inscription d'un nouveau service : il peut soit être placé dans une catégorie existante, soit il faut créer une nouvelle catégorie adaptée à ce nouveau type de service.

Pour éviter ce problème, la classification ne peut être indépendante du catalogue de service mais doit se baser sur les descriptions de service stockées dans le catalogue de service. Le catalogue gérera également la hiérarchisation des différents axes de classification.

La classification sera donc générée à la demande de l'utilisateur sur base d'une liste de services et d'un type de classification. Les liens entre notre catalogue et la classification sont

15. Un alignement pairwise fait référence à un alignement de deux séquences.
16. Il n'est pas nécessaire de donner trop de détails à un débutant.

exprimés dans la figure 5.12. Nous avons représenté en bleu les éléments non permanents.

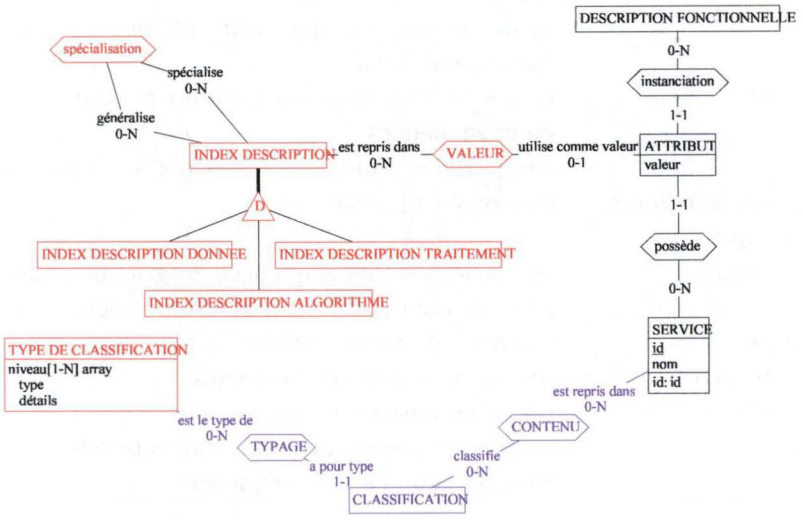


FIG. 5.12 – *Modélisation de la classification*

L'information sur le traitement et l'algorithme est déjà présente dans la description fonctionnelle. Nous rajoutons un attribut de description fonctionnelle pour les données, cet attribut sera lié à la signature pour préserver sa sémantique.

5.4.2 Service abstrait

Une tâche peut être réalisée par différents algorithmes, ainsi différents algorithmes permettent de trier un tableau comme le tri à bulles, le tri par insertion, etc. Ces différents algorithmes ont des caractéristiques comme le temps d'exécution. Le programmeur choisira la technique à utiliser d'après différents facteurs comme la taille des données à traiter.

L'idée est la même en bioinformatique. Un service permet de réaliser une tâche au moyen d'un algorithme. Le biologiste doit trouver le service le mieux adapté selon une série de facteurs comme la taille de sa donnée.

Mais en bioinformatique, la tâche sert de base à la classification. En effet, la difficulté de concevoir des outils permettant de réaliser et de visualiser une tâche biologique, a conduit les programmeurs à spécifier des tâches plus simples ne reprenant qu'une partie de la tâche biologique. Pour choisir ces tâches, le biologiste se basera également sur des facteurs comme le nombre de données.

La figure 5.13 reprend les critères de choix pour la tâche alignement.

Méthode	facteurs influençant le choix
alignement	
pairwise	compare deux séquences Si on ne sait pas quel type utiliser toujours utiliser un alignement local.
dot plot	représente sur un graphique les régions de similarité des deux séquences
local	compare les séquences sur la région la plus similaire
smith-waterman	très précis et donc lent
water	
matcher	optimisé pour les séquences de grande taille propose comme résultat plusieurs régions similaires
blast	rapide mais moins précis
blastp	prend en entrée des protéines
blastn	prend en entrée des nucléotides
global	compare les séquences sur leur entièreté
multiple	compare une liste de séquences

FIG. 5.13 – *Choix d'une méthode d'alignement*

Remarque : seul water, matcher, blast, blastp et blastn sont des services.

Aujourd'hui, des informations du type du tableau présenté à la figure 5.13, autrement dit les informations permettant à l'utilisateur de choisir le service, ne sont pas apparentes. Le biologiste trouvera principalement cette information dans des manuels de bioinformatique (par exemple [Claverie et Notredame, 2003]) ou dans l'aide des programmes.

Nous voulons donner à l'utilisateur un accès clair à cette information. Pour ce faire, nous avons ajouté cette information sous la forme de deux attributs : *nom et description* dans l'entité *index description* (voir la classification, figure 5.12).

En complément, nous pensons créer des services abstraits. Un service abstrait correspond à une tâche et réalise avec l'aide de l'utilisateur le choix du meilleur service concret [17] d'après les données de l'utilisateur. Ce service abstrait est une alternative à l'utilisation par certains biologistes d'un seul programme répondant tant bien que mal à leurs besoins.

De plus, le biologiste utilise peu l'ordinateur, en général, une fois tous les six mois pour préparer une expérience, analyser certains résultats, etc. On ne peut donc pas lui demander de retenir énormément d'informations. Or, seule l'information nécessaire à la réalisation de sa tâche sera nécessaire pour utiliser un service abstrait.

Un service abstrait ne peut être instancié mais il est caractérisé par une signature et une description. La figure 5.14 représente notre catalogue de service amélioré. Les éléments en vert représentent les caractéristiques de tous les services et les éléments en bleu représentent les caractéristiques des services concrets.

Les services abstraits se situeront plus haut dans la classification que les services concrets car les services abstraits sont moins caractérisés que leurs homologues concrets. Comme

17. Un service concret est un service actuel, lié à une implémentation.

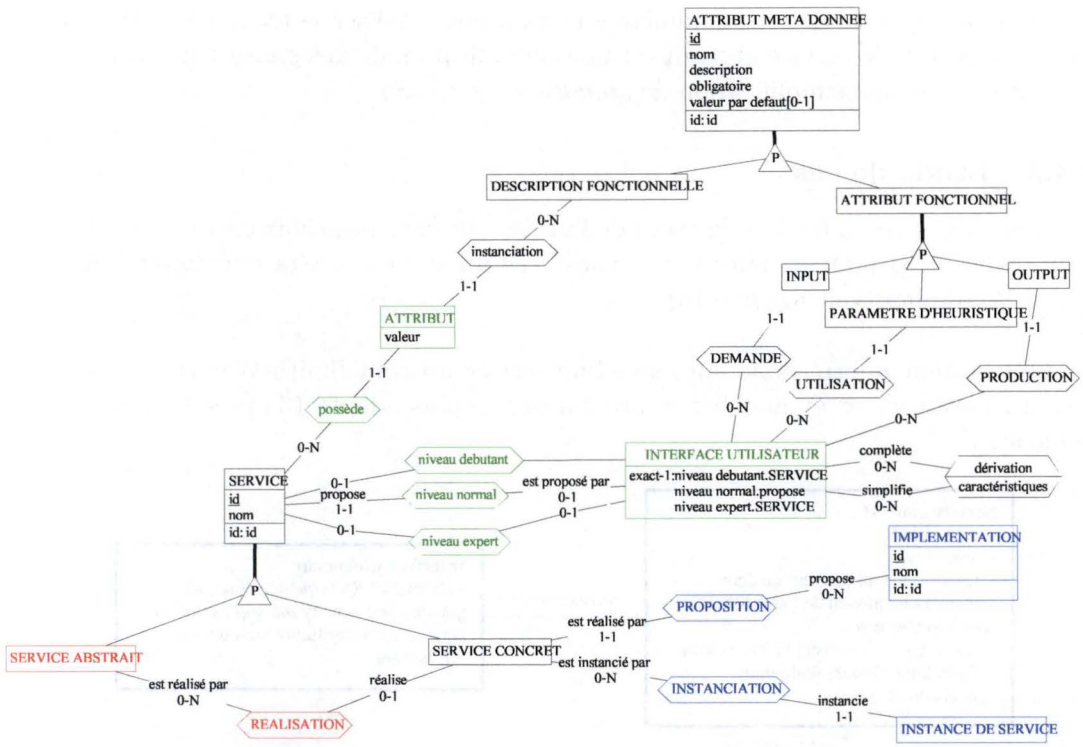


FIG. 5.14 – *Modélisation des services abstraits*

plus proche du haut de la classifications, ces services seront plus accessibles aux utilisateurs débutants. Dans cette optique, nous voulons intégrer dans ces services des informations sur les bonnes pratiques. Ainsi, le service *alignement pairwise* permettra à l'utilisateur de réaliser un *dot plot* et un *alignement pairwise local* car ces deux programmes apportent des informations complémentaires.

Les caractéristiques d'un service abstrait (comme la tâche) ne sont pas indépendantes des caractéristiques des services concrets réalisant ce service abstrait. En effet, le service abstrait s'appuie sur les services concrets.

Soit la définition suivante, soit $S(t)$ = ensemble des services réalisant le traitement t,

$$description_t = \bigcap_{x \in S} description_x$$

Par exemple, la description d'un alignement reprendra toutes les caractéristiques des services permettant la réalisation de cet alignement [18].

Les liens entre signatures sont plus complexes. Certains services abstraits seront tellement éloignés des services concrets qu'il sera impossible de trouver des liens.

Seul le service abstrait représentant un type d'algorithme peut nous apporter de l'information. En effet, le type d'algorithme donne de l'information sur le différentes étapes subies

18. alignement pairwise local ∩ alignement pairwise dot-plot = alignement pairwise

par les données ainsi que les paramètres utilisés pour réaliser le traitement. Donc dans ce cas, la signature du service abstrait est une simplification de la signature du service concret ($signature_s$ est une simplification de $signature_x \; \forall \; x \in S$).

5.4.3 Etude de cas

Nous avons voulu faciliter la tâche de l'utilisateur dans son choix entre water et matcher (voir table 5.13) pour ce faire nous avons créé un service abstrait réalisant l'algorithme Smith-Waterman (voir figure 5.15).

L'utilisateur pourra donc faire appel au service abstrait Smith-Waterman. Le système choisira parmi water et matcher le programme le plus adapté (d'après la longueur de la séquence).

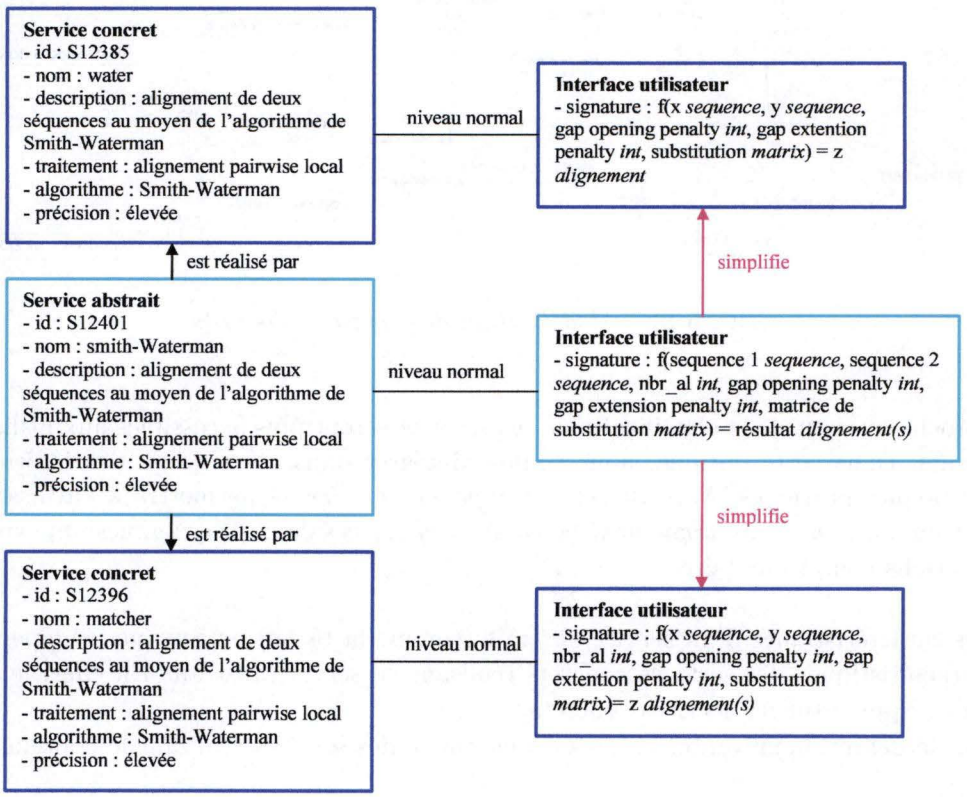


FIG. 5.15 – *Modélisation du service abstrait : Smith-Waterman (application)*

5.5 Catalogue consolidé

Nous avons réalisé dans ce chapitre un schéma global (voir figure 5.16) du catalogue de services. Chaque acteur est intéressé par une vue précise de ce catalogue.

Par exemple, l'utilisateur lors de sa recherche de service est intéressé par l'information sur le service : sa classification, sa description fonctionnelle (en ce compris le traitement

et l'outil métier utilisé), les attributs non fonctionnels et enfin la signature. Lors de la paramétrisation du service, il sera intéressé par la signature du service. Enfin a posteriori, il aura besoin d'accéder à l'historique de ses appels pour obtenir de l'information sur une donnée, effectuer un rejeu, etc.

5.6 Conclusion

Dans ce chapitre, nous avons d'abord analysé des techniques utilisées aujourd'hui pour modéliser les services de bioinformatique, plus particulièrement le langage ACD.

Nous avons ensuite réalisé une modélisation des services. Celle-ci permet de modéliser tout d'abord notre catalogue de services. Nous l'avons ensuite étendue avec des concepts propres à la bioinformatique : les outils métier et les traitements métier. Enfin, nous avons présenté l'ensemble de notre modélisation.

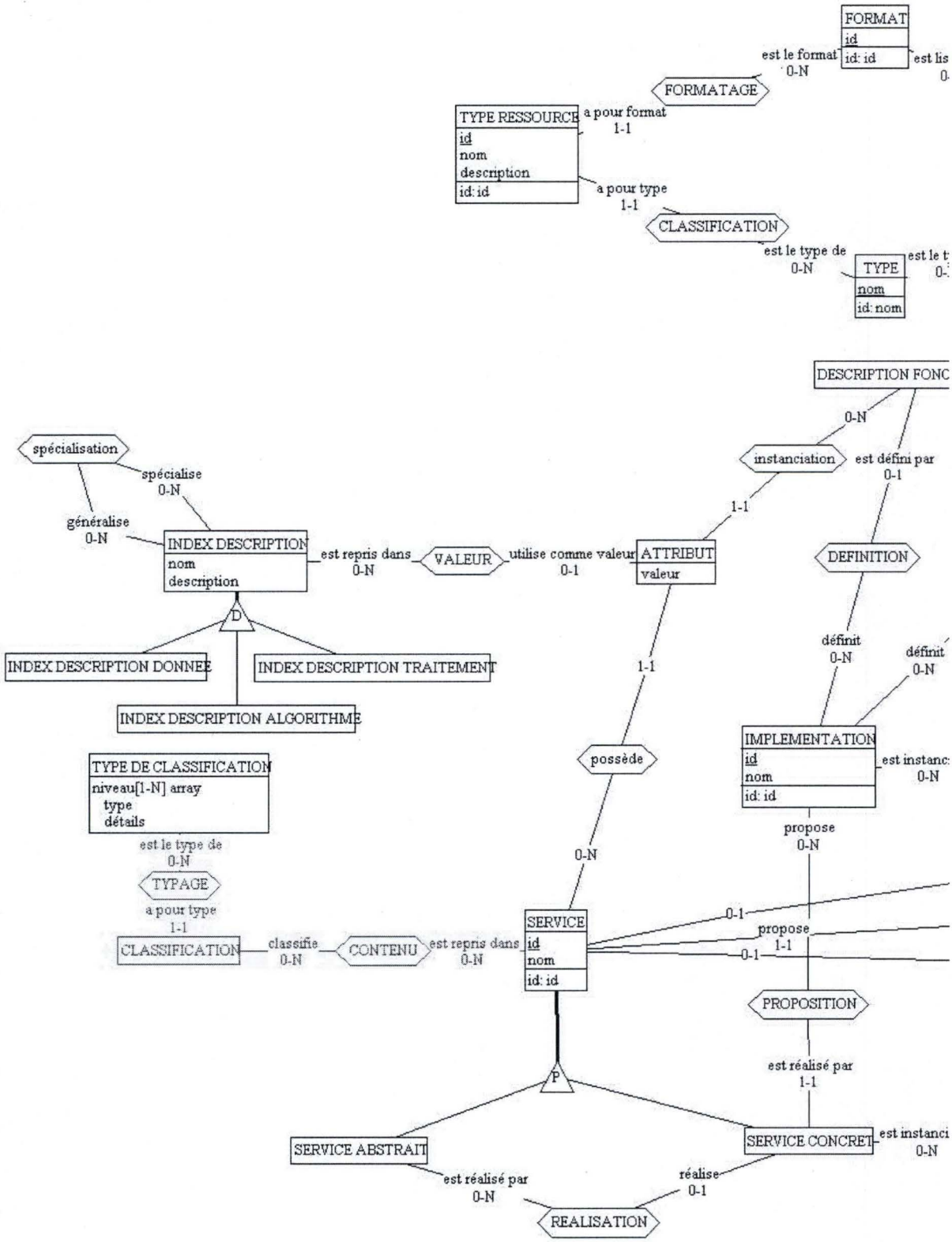


FIG. 5.16 – *Catalogue de services : vue complète*

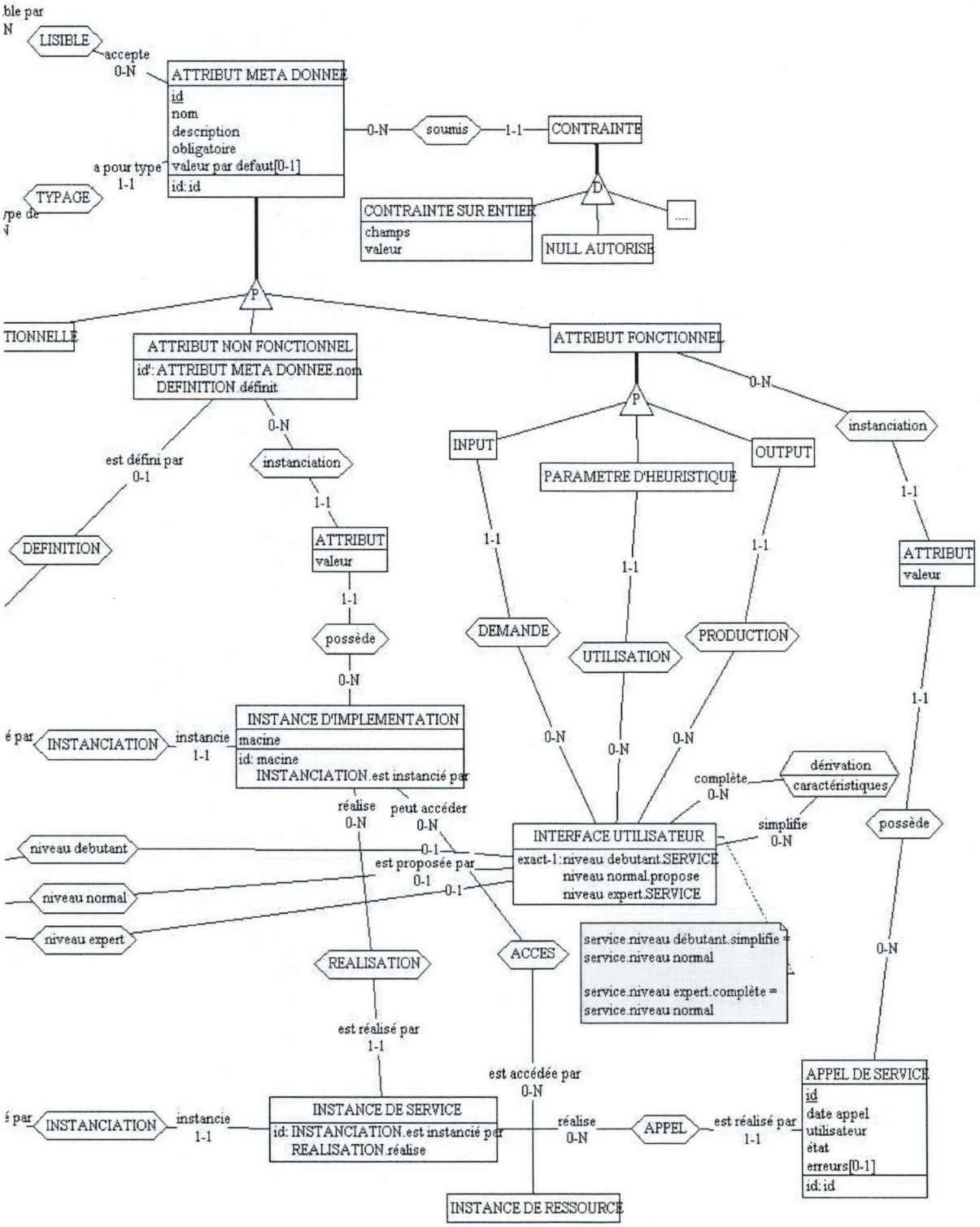


Figure 5.16 (suite)

Chapitre 6

Modélisation de l'Interface Homme Machine

Aujourd'hui, les fournisseurs proposent aux utilisateurs une interface (généralement une page web) permettant l'utilisation de programmes bioinformatiques. Ils ont réalisé eux-mêmes l'interface[1], ou ont utilisé des interfaces réalisées par un tiers : autre fournisseur, auteur, etc.

Or, dans notre modélisation, l'interface ne dépend plus du fournisseur, le catalogue de service contient pour chaque service une interface (ou *description IHM*). Cependant, il nous semble important de laisser aux fournisseurs qui le désirent la possibilité de réaliser leur propre interface. Pour ce faire, il est nécessaire de modéliser l'IHM et d'ainsi décrire les éléments que le fournisseur doit réaliser.

Pour réaliser notre modélisation, nous allons nous baser sur l'architecture fonctionnelle [Bodard, 2002]. Cette architecture est basée sur la séparation en trois types de composants : présentation, conversation et fonctionnel.

Cette séparation permet de mettre en évidence les flux de communication entre ces différents composants (voir figure 6.1).

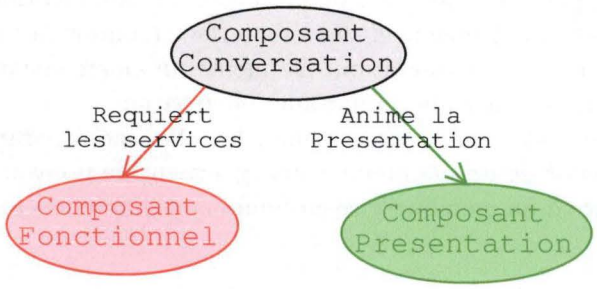


FIG. 6.1 – *Séparation des différents composants d'IHM [Bodard, 2002]*

1. L'IBMM a réalisé pour ses utilisateurs une interface : wEMBOSS [Sarachu et Colet, 2004]. Ce logiciel open-source disponible sous la licence GNU (General Public License) est une interface web au package de bioinformatique EMBOSS.

Pour comprendre ces différents composants, nous allons analyser leurs fonctions :

1. Le **composant conversation** à un double rôle :
   - gérer le dialogue (la conversation) avec l'utilisateur en assurant la gestion du **composant présentation** qu'il anime
   - requérir les services de l'application via des appels aux primitives du **composant fonctionnel**.

   Ce composant joue donc un rôle médiateur entre **l'interface** qu'il contrôle et anime et le **composant fonctionnel** qu'il sollicite.

2. Le rôle du **composant présentation** est d'organiser la disposition des objets de la présentation en vue de permettre à l'utilisateur de réaliser les actions associées à sa tâche.

   Ce composant est constitué d'un ensemble d'éléments interactifs qui permettent à l'utilisateur de réaliser les actions :
   - de saisie ou d'affichage d'informations : champ d'édition, etc.;
   - de lancement de commande : bouton de commande, menu, etc.;
   - de prendre connaissance de messages d'erreurs, de progression, etc.

   Dans le paradigme orienté objet, chacun de ces éléments :
   - réagit aux sollicitations des utilisateurs par la génération d'événements qui sont destinés à signaler tout changement d'état intervenu au niveau de la présentation. Ces événements sont "reçus" par d'autres composants, en l'occurrence le **composant conversation** dans notre architecture, qui réagit alors en conséquence. (sélection d'un bouton, frappe d'un nouveau caractère, etc.)
   - offre un ensemble de primitives que les **autres composants** peuvent appeler. (obtenir la valeur d'un champs de texte, modifier cette valeur, activer un bouton, etc.)

3. Le **composant fonctionnel** met à disposition du **composant conversation** tous les services (les fonctions) que ce composant souhaite solliciter.

Cette architecture permet de localiser plus aisément les modifications destinées à d'éventuelles améliorations. En effet, un changement dans l'interface (comme la taille, la couleur) affectera seulement le composant présentation. De même un changement dans l'enchaînement des fenêtres[2] affectera seulement le composant conversation.

Cette architecture nous semble intéressante. En effet, en séparant ces éléments, il est non seulement plus facile de les maintenir mais également de pouvoir disposer de plusieurs versions d'une interface d'un même service en combinant différentes présentations et conversations.

Pour modéliser une interface de service bioinformatique, les aspects **composant présentation** (apparence) et **composant conversation** (interactions avec l'utilisateur) nous semblent les points clés de cette modélisation d'IHM. En effet, on peut considérer le **composant fonctionnel** comme standard à tous les services bioinformatiques. Il comprendrait par exemple

2. Le changement dans l'enchaînement n'est possible que si la logique de l'application l'autorise. Il n'aurait aucun sens de visualiser un résultat sans avoir demandé l'exécution du service qui produira ce résultat.

des fonctions de recherche et d'appel de service, etc.

Nous étudierons d'abord le composant présentation, puis le composant conversation. Ensuite, nous proposerons des outils pour réaliser cette modélisation et ainsi permettre à l'utilisateur d'interagir avec un service. Enfin, nous mettrons ce chapitre en rapport avec notre modélisation du service.

6.1 Présentation

Aujourd'hui, le service se présente comme un formulaire permettant à l'utilisateur de compléter différents paramètres et de lancer ensuite l'exécution du service. Forcément, les interfaces actuelles permettent également à l'utilisateur de visualiser les résultats de ces exécutions.

Dans notre modélisation, la présentation de données, de services sera réalisée par un visualisateur. Celui-ci permettra de représenter entre autre le formulaire actuel (dans le cas des services) et sera composé des différents éléments d'IHM : champ de saisie, bouton, etc. avec lesquels l'utilisateur pourra ensuite dialoguer.

Néanmoins, la construction d'un visualisateur ne doit pas être réalisée "from scratch", cela prendrait trop de temps. Il faut donc disposer d'outils permettant la réalisation de ces visualisateurs.

Le type de la donnée va influencer la présentation. En effet, la présentation d'un paramètre de type entier nécessite un champ de saisie n'acceptant que des entiers. Par contre, la présentation d'une protéine requiert un champ de saisie plus grand (pouvant contenir 500 caractères) qui ne doit permettre que les caractères correspondants aux différents acides aminés.

Les outils pour aider à la création du `visualisateur` seront donc basés sur la notion de "`type`". On peut donc modéliser la présentation par la figure 6.2.

Le `modèle` est la mise en évidence de la présentation d'un `type`, il est caractérisé par les différents éléments d'IHM : champs de saisie, bouton, etc. Par exemple, le `modèle` d'un entier sera un label reprenant le nom d'un paramètre et un champ permettant d'encoder la valeur de ce paramètre. Néanmoins, aucune information sur le nom ou la valeur de la donnée n'est encodée mais leur place est définie.

Pour former un boîte à outils suffisamment complète, chaque `type` peut être visualisé à l'aide d'au moins un `modèle`. Plusieurs `modèles` peuvent coexister pour un même `type` car certaines données nécessitent parfois un rendu travaillé.

Un visualisateur sera donc construit avec des `modèles`. Or celui-ci ne contient pas les informations suivantes : nom du service, valeur de la donnée, etc. . Ces informations sont stockées dans l'entité `contenu` qui instanciera le `modèle` pour en faire un `visualisateur`.

Forcément, les *informations contenues* sur la donnée et sur le service sont très différentes. Une donnée est caractérisée par un nom et/ou une valeur. Les services eux sont caractérisés par des paramètres. Or ces paramètres sont surtout des informations (nom de ce paramètre, valeur éventuelle) qui ne correspondent à aucune interface. Pour être présentés à l'utilisateur, les paramètres doivent donc prendre la forme d'un `visualisateur`.

Nous allons décrire en détail ces `visualisateurs` dans les sections suivantes.

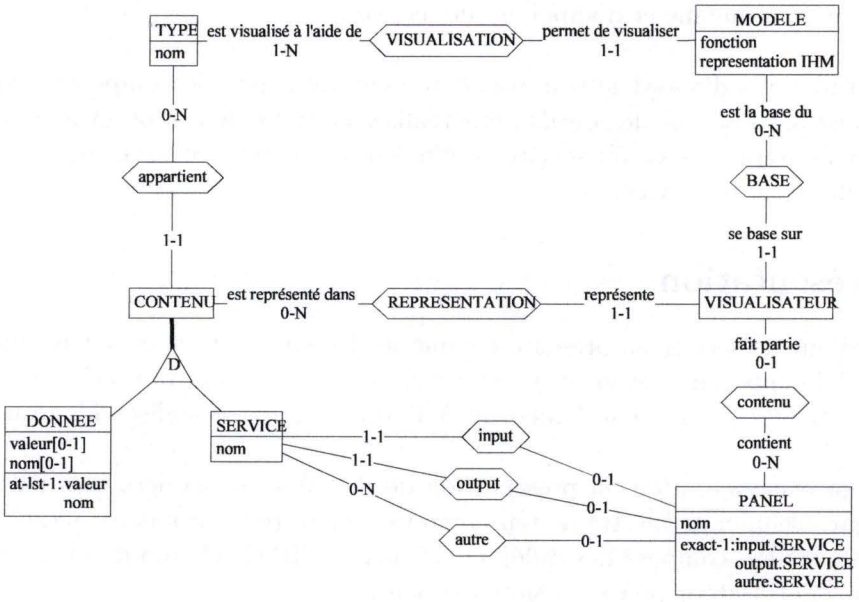


FIG. 6.2 – ***Modélisation de la présentation*** *La donnée/le service est visualisé au moyen d'un visualisateur. Ce visualisateur est construit à partir d'un modèle (présentation d'un type de données) et des informations spécifiques (contenu) sur la donnée ou le service.*

6.1.1 Visualisateur de donnée

Le `modèle` présente donc les différents éléments d'IHM permettant d'afficher cette donnée mais ni le nom, ni la valeur de la donnée ne sont connus. Leurs emplacements sont simplement définis.

Comme toute présentation, le `modèle` propose une interface permettant à l'utilisateur de réaliser une tâche. La fonction du modèle exprime la tâche pour lequel on a réalisé cette présentation : paramétrer (édition), visualiser, valider, etc.

Par exemple, la figure 6.3 est un modèle qui permet à l'utilisateur de compléter un formulaire au moyen d'une donnée de type séquence. Les informations manquantes sont présentées en rouge.

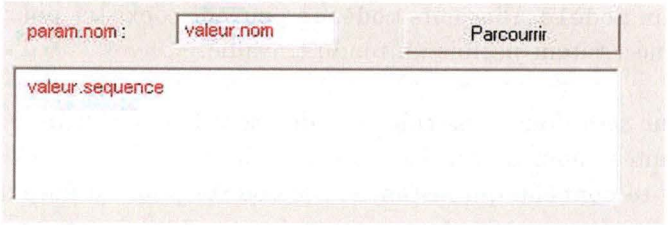


FIG. 6.3 – *Modèle permettant de visualiser le type séquence*

Pour obtenir un `visualisateur` qui sera affiché, il faut ajouter au `modèle` de la figure 6.3 un nom de paramètre ainsi qu'éventuellement une valeur. Ces valeurs remplaceront les

éléments en rouge : le nom du paramètre, le nom et la chaîne de caractères correspondant à la valeur de la séquence.

Une fois affiché, l'utilisateur aura la possibilité de remplir les valeurs prises par le paramètre : nom et chaîne de caractères.

Un même type peut être visualisé au moyen de plusieurs modèles pour permettre de visualiser les différentes facettes d'une donnée mais également pour refléter la tâche à réaliser.

Par exemple, le paramètre x (nom = "param-3" et valeur = 5) peut être visualisée par deux modèles (voir figure 6.4). Les deux modèles utilisés servent à la paramétrisation. Le premier donne une grande liberté à l'utilisateur dans le choix de la valeur. Le second insiste sur le concept de distance entre la valeur initiale (généralement la valeur par défaut) et la future valeur choisie par l'utilisateur.

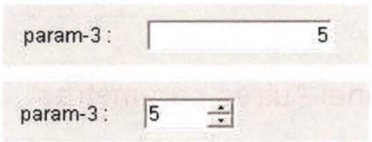


FIG. 6.4 – *Visualisateurs du paramètre x de type entier*

6.1.2 Visualisateur de service

La définition du service classait les paramètres en trois catégories : input, output et autres paramètres. Il nous semble essentiel que cette distinction se retrouve dans l'Interface Homme Machine. En effet, cette distinction avait été réalisée pour correspondre à la vue qu'avait l'utilisateur du service.

Le service peut donc être représenté par trois groupes de paramètres. Les groupes *input* et *output* sont obligatoires. Le groupe *autres paramètres* est optionnel et peut encore être découpé en sous-catégories, d'après les éventuels besoins.

Nous allons représenter ces groupes par des panels. Un panel comprend les visualisateurs des différents paramètres, c-à-d à la fois la présentation (modèle) et le contenu.

Le modèle de service présente les différents éléments d'apparence comme les boutons permettant de lancer l'exécution et il prévoit surtout un emplacement pour les différents panels.

Par exemple, la figure 6.5 est un modèle de service. C'est une présentation classique en bioinformatique. Elle permet à l'utilisateur de compléter les paramètres essentiels du service (input et output). Lorsque l'utilisateur a paramétré le service, il peut demander l'exécution du service. En cas de besoin, il peut accéder directement à l'aide du service.

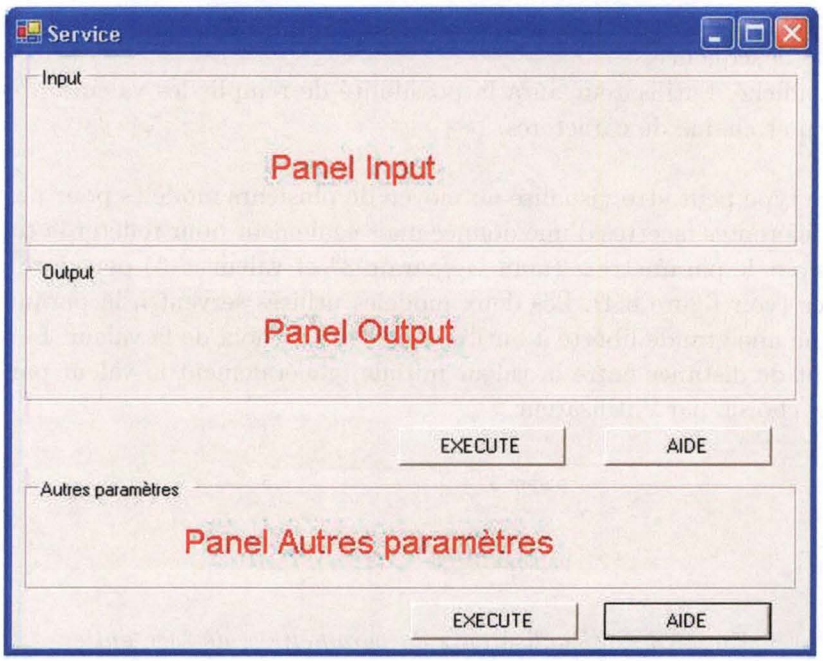


FIG. 6.5 – *Modèle de service*

A partir d'un modèle, nous pouvons réaliser un visualisateur pour n'importe quel service, par exemple le service *algorithme Smith-Waterman* défini dans le chapitre précédent (voir figure 5.15 page 66). Les informations importantes du service sont : un nom ainsi qu'une signature qu'il faut transformer en panels.

La signature met en évidence les données que l'utilisateur peut compléter afin de réaliser un appel de service. L'information présente dans la signature permet donc de connaître le contenu des différents paramètres. Cette information sera présente dans les panels (voir table de la figure 6.6). Chaque type de paramètres (input, output et autres paramètres) se retrouve dans le même panel[3].

Ensuite, pour chaque paramètre, il faut choisir le modèle qui permet de présenter le mieux le paramètre. Comme le premier paramètre est une séquence, nous avons choisi la figure 6.3 qui présente une séquence à paramétrer. La seconde colonne de la table de la figure 6.6 fait référence aux modèles choisis.

Pour constituer le visualisateur du service, il suffit d'associer le modèle et le contenu. Nous obtenons ainsi la figure 6.7.

6.1.3 Présentation d'autres services

La visualisation des données et des services ne suffit pas à caractériser toutes les présentations que le système devra afficher. En effet, il faudra également prévoir la possibilité de décrire d'autres interfaces comme l'aide d'un service, les fenêtres basiques permettant d'ouvrir un fichier, etc.

3. Nous n'avons pas subdivisé la partie "autres paramètres".

Paramètres	Modèle
Input	
sequence 1	voir figure 6.3
sequence 2	voir figure 6.3
Output	
résultat	modèle permettant de visualiser le type *alignement(s)*
Autres paramètres	
gap opening penalty	premier modèle, voir figure 6.4
valeur = 5	
gap extension penalty	premier modèle, voir figure 6.4
valeur = 3	
matrice de substitution	modèle permettant de visualiser le
valeur = BLOSSUM62	type *matrix*

FIG. 6.6 – *Le contenu : données du service* algorithme Smith-Waterman

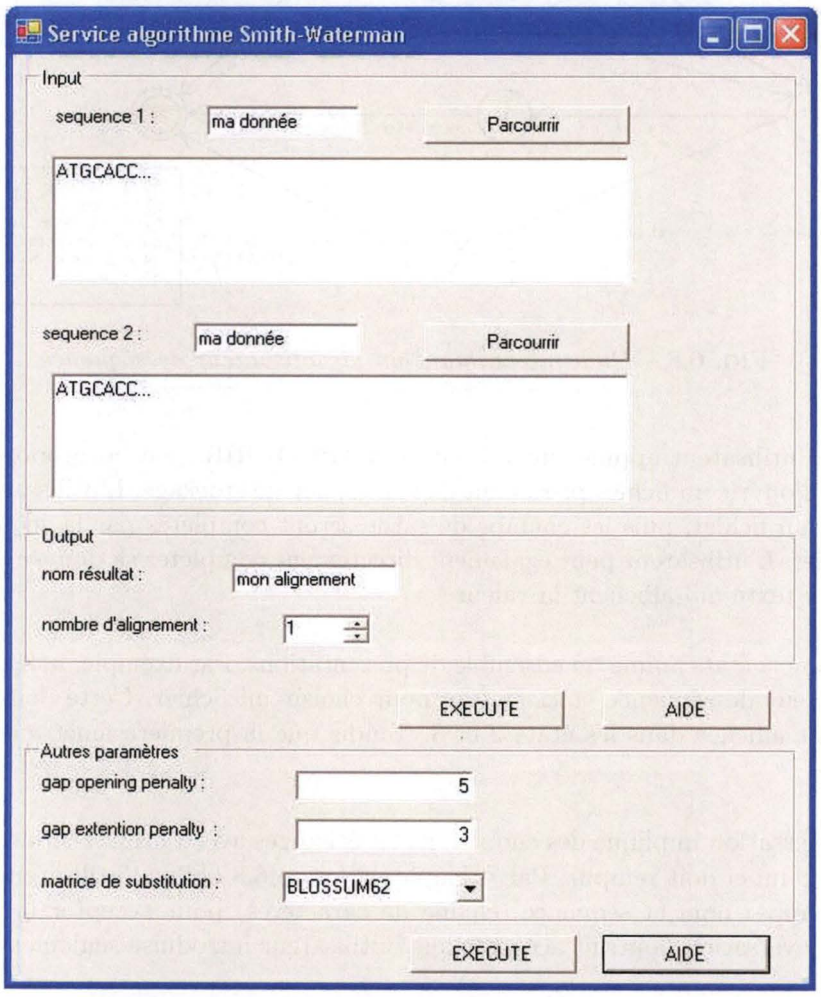


FIG. 6.7 – *Visualisateur du service* Smith-Waterman

6.2 Conversation

La conversation sera vue comme un dialogue entre l'utilisateur et l'ordinateur au moyen de la présentation. Ce dernier entend l'utilisateur par l'écoute des événements et réagit pour adapter la présentation et faire des appels au composant fonctionnel.

Pour spécifier la gestion du dialogue, nous allons utiliser une machine à états où une transition est caractérisée par l'événement qui la "déclenche" et par des appels de primitives dont l'exécution permet de passer de l'état de départ à l'état d'arrivée.

La figure 6.8 représente la conversation d'un visualisateur de séquence. Ce visualisateur se base sur le modèle présenté à la figure 6.3. La transition est caractérisée par l'événement qui a déclenché la transition (en mauve) et par l'action qui est exécutée en réponse (en vert).

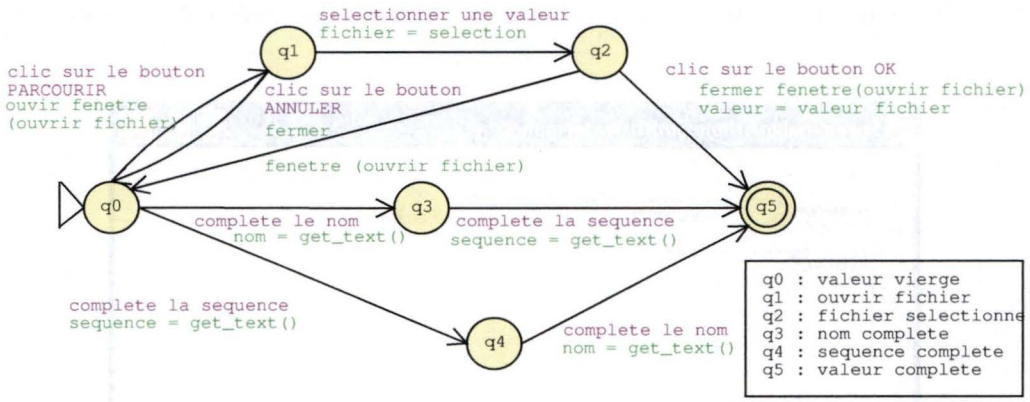


FIG. 6.8 – *Dialogue animant un visualisuateur de séquence*

Lorsque l'utilisateur appuie sur le bouton "PARCOURIR", on lui propose une fenêtre permettant d'ouvrir un fichier provenant de son espace de stockage. L'utilisateur peut ainsi sélectionner un fichier, puis les champs de saisie seront complétés par la séquence stockée dans ce fichier. L'utilisateur peut également directement compléter sa donnée au moyen des deux champs texte qui affichent la valeur.

La machine à états anime un ensemble de présentations. Par exemple, la figure 6.8 anime un visualisateur de séquence et la fenêtre pour choisir un fichier. Cette deuxième fenêtre est seulement affichée dans les états 2 et 3. Tandis que la première fenêtre est affichée en permanence.

Une conversation implique des choix dans les échanges avec l'utilisateur et dans les obligations que celui-ci doit remplir. Par exemple, la figure 6.8 oblige l'utilisateur à compléter les deux champs : nom et séquence (chaîne de caractères) pour accepter la valeur. Mais, une autre conversation pourrait accepter que l'utilisateur introduise seulement la chaîne de caractères.

La figure 6.9 représente la méta-modélisation de notre machine à états, ainsi que les liens avec la présentation. Les concepts de la machine à états, représentés en bleu, sont basés sur

la modélisation *statechart* définie par le langage UML.

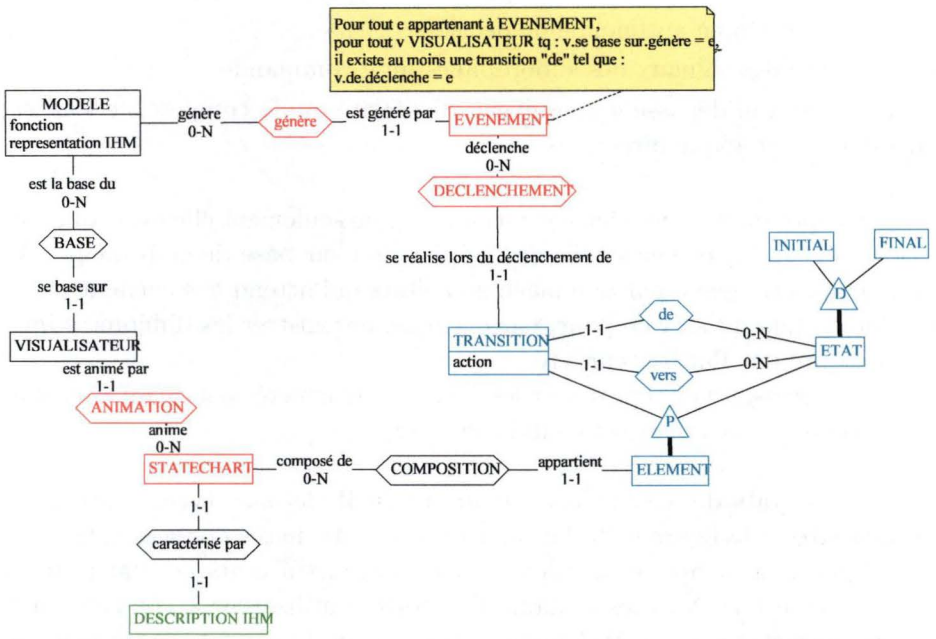


FIG. 6.9 – *Modélisation du dialogue*

6.2.1 Multi-conversation

Le visualisateur du service est composé d'un ensemble de visualisateurs de données. Or chacun de ces visualisateurs est animé par sa propre machine à états. La conversation du service comprend donc sa conversation ainsi que les différentes conversations menées par ses différents paramètres.

La machine à états du service doit disposer d'un mécanisme lui permettant de gérer, contrôler les différentes "conversations données" car les interactions de l'utilisateur avec un paramètre peuvent avoir des répercutions sur le dialogue du client avec le service.

Par exemple, l'utilisateur ne peut exécuter le service que lorsque tous les paramètres ont une valeur. Pour réaliser cette contrainte, le bouton EXECUTER ne sera activé que lorsque toutes les machines à états des données seront dans un état final.

Une première solution serait de déployer les différentes machines à états des données dans la machine à états du service. Autrement dit, intégrer tous les états des différentes "conversations donnée" dans la "conversation service".

Cette solution n'est pas évolutive. En effet, un changement dans une machine à états d'une donnée devra être répercuté dans la machine à états du service.

Une deuxième solution consiste à demander aux machines à états des données de générer des événements. Ces événements seront récupérés par la machine à états du service et qui pourra ainsi agir en conséquence.

Cette solution [Bodard, 2002] se base sur l'organisation d'une entreprise hiérarchisée en différents niveaux d'autorité et de compétence. Dans ce type d'organisation chaque individu

- possède une certaine autonomie de décision
- est à l'écoute des signaux des subordonnés qu'il commande
- les problèmes qui dépassent son niveau d'autorité ou de compétence sont signalés au supérieur hiérarchique direct.

Nous avons opté pour la deuxième solution, car non seulement elle est évolutive mais elle permet également de gérer l'ensemble de l'application sur base de ce principe. Ainsi, l'application pourrait être gérée par une machine à états qui attend des événements provenant de la machine à états du service pour, par exemple, enregistrer les différentes informations sur l'appel exécuté par l'utilisateur, etc.

Mais il sera nécessaire de répertorier les événements générés par chaque machine à états comme par exemple, "la valeur est complétée", etc.

La machine à états du service *algorithme Smith-Waterman* (visualisateur, voir figure 6.7) est représentée à la figure 6.10. Les machines à états de ses données envoient un signal lorsque l'utilisateur a complété la valeur (ou la machine à états est dans un état final). Lorsque le service a reçu tous les signaux, il autorise l'utilisateur à appuyer sur le bouton **exécuter** lui permettant ainsi de demander l'exécution du service. Cette machine permet également à l'utilisateur d'accéder en permanence à un fichier d'aide.

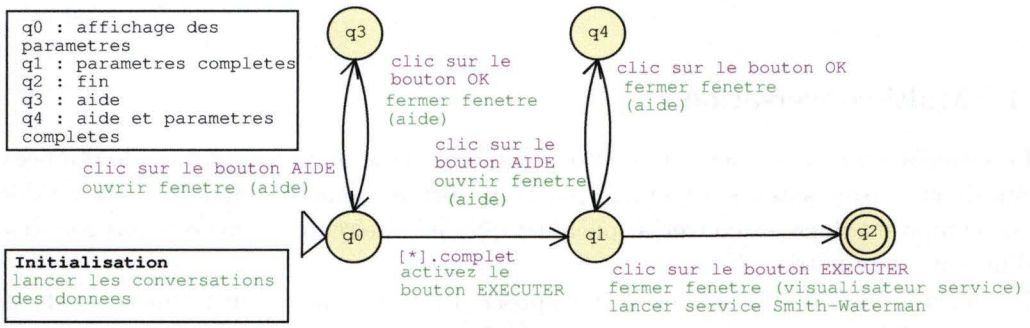


FIG. 6.10 – *Dialogue animant un visualisateur de service*

Notre solution est caractérisée par les propriétés suivantes:

- la **circulation de signaux** (événements) entre éléments n'est possible qu'entre deux éléments liés par un lien de hiérarchie direct. Pour traverser tout ou partie d'une hiérarchie, un signal doit transiter par le supérieur hiérarchique direct.
- l'**intelligence** de l'organisation est distribuée, car chaque élément "règle" au mieux les problèmes qui lui sont posés. Si ces problèmes dépassent son niveau d'autorité ou de compétence ils sont reportés vers le supérieur direct.
- la **structuration des communications**, par les limitations qu'elle impose, permet de réduire deux risques:
  - celui de **court-circuiter un niveau**: le fait qu'un composant travaille à l'insu de son supérieur direct

- et celui de **noyer les composants** de niveaux supérieurs sous une pluie de signaux dont ils n'ont que faire. Chaque composant doit assumer ses propres responsabilités et ne reporter vers le niveau supérieur que les problèmes qu'ils ne peut résoudre à son niveau.

Autres conversations. Certaines conversations ne sont pas signalées dans la machine à états comme la gestion des contraintes,la gestion de la présentation[4], etc. En effet, ces éléments n'apportent pas énormément d'informations sur la conversation et pourraient embrouiller les schémas. De plus, ils sont communs à tous les services.

6.3 Réalisation

Nous avons une modélisation permettant au fournisseur de réaliser des interfaces. Du côté client, le travail déjà réalisé par les chercheurs de Bigre a mis en évidence la nécessité d'une architecture client modulable qui s'adapte au service invoqué au travers de téléchargements des descriptions de service. La description d'IHM (présentation et conversation) doit donc être stockée puis téléchargée et enfin visualisée par l'utilisateur.

Nous allons maintenant examiner un outil capable de gérer ces contraintes : XUL (XML User Interface Language). Puis nous étudierons comment nous pouvons l'utiliser.

6.3.1 Technique : XUL

XUL est un langage de représentation des interfaces en XML [Andersen et Deakin, 2004]. Ce langage a été créé pour rendre le développement de l'interface du navigateur Mozilla plus facile et plus rapide. D'ailleurs, aujourd'hui, l'interface de ce navigateur est entièrement réalisée dans le langage XUL.

Ce langage a été conçu pour être portable et est disponible sur toutes les versions de Windows, Macintosh, Linux ainsi que UNIX.

De plus, les éléments ont été conçus pour avoir l'apparence des éléments natifs de la plate-forme de l'utilisateur. Il est également possible de changer cette apparence au moyen des feuilles de style.

Comme l'HTML, XUL permet de créer une interface en utilisant des étiquettes[5]. Il est donc aisé de localiser les éléments comme un texte affiché. De plus, il est aisé de traduire une interface XUL vers un autre langage. Enfin, le langage JavaScript est utilisé pour coder la conversation. Certaines librairies sont déjà définies pour lire et écrire des fichiers distants, faire appel à des webs services et lire des fichiers locaux.

Contrairement à l'HTML, XUL fournit plus d'outils permettant la construction d'IHM comme les menus, les barres à outils, les onglets et les arbres, etc. En plus de ces nombreux widgets[6] disponibles, le langage : eXtensible Bindings Language (XBL) permet de créer des

4. La gestion de la présentation est la gestion d'événements ayant des impacts sur la présentation des éléments, par exemple, l'apparence d'un bouton qui change lorsque la souris se trouve dessus.

5. Une étiquette (ou tag en anglais) est un morceau de texte utilisé en XML, HTML pour représenter le début ou la fin d'un élément. Une étiquette commence par < et termine par >.

6. Un widget est un élément de présentation comme un champ de saisie.

widgets supplémentaires. Ce langage peut être employé pour créer de nouvelles étiquettes et pour implémenter de nouvelles fonctionnalités.

L'interface est stockée dans un fichier (appelé package). Celui-ci est divisé en trois parties :

Content - fenêtre et script.
Les déclarations des fenêtres sont stockées dans des fichiers .xul. Les scripts son placés dans des fichiers séparés.

Skin - feuilles de style, image et autre fichier spécifique au style.
La feuille de style décrit les détails de l'apparence d'une fenêtre. Ces éléments sont stockés séparément pour pouvoir facilement changer le thème (apparence) d'une application. Les images utilisées sont également stockées dans cette partie.

Locale - fichiers spécifiques aux régions.
Tous les textes affichés dans les fenêtres sont stockés dans un fichier. Cela permet à l'utilisateur d'utiliser l'interface dans son propre langage.

Un package ne comprend pas forcément toutes ces différentes parties.

Les applications peuvent être soit visualisées par l'utilisateur depuis un site web, soit téléchargées par l'utilisateur puis installées comme une extension de Mozilla (voir ci-après). Installer une application permet à l'interface de posséder plus de permissions comme par exemple, lire les préférences de l'utilisateur, les informations sur son système, etc.

XUL permet d'ajouter facilement des nouveaux packages qui seront considérés comme des extensions. Ces packages seront installés dans Mozilla et l'utilisateur aura l'impression que cet ajout de code est intégré dans l'interface et donc que ce bout de code est indissociable du logiciel Mozilla mais en réalité, ce code est toujours séparé du code initial. Il est donc aisé de désinstaller une extension.

L'extension pourra permettre, par exemple, la création d'une barre d'outils personnalisée, le changement des menus, etc. Cette caractéristique est fort utilisée pour le navigateur Mozilla Firefox où environ 100 extensions sont disponibles comme une barre d'outils permettant un accès rapide au moteur de recherche Google ainsi que des outils annexes (recherche dans le site visité,..). Certaines fonctionnalités de cette extension sont disponibles à partir d'un menu.

Actuellement, des recherches sont en cours pour permettre à des applications XUL d'être créées comme applications autonomes avec leurs propres installateurs et exécutables. Ces applications partageront des librairies avec Mozilla mais l'utilisateur n'aura plus besoin d'un navigateur installé pour employer XUL.

XUL répond à nos objectifs en terme d'adaptation aux différentes plate-formes existantes à l'heure actuelle ainsi qu'à la facilité de création, d'ajout d'éléments et de changement de style.

Concrètement une interface se présente comme un document XML (voir figure 6.11). Les éléments de présentation sont représentés par des balises avec des attributs comme le nom, la valeur, etc. Ce document peut être visualisé au moyen de Mozilla pour donner cette interface (voir figure 6.12).

```
<?xml version="1.0" ?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css" ?>

<window id="example-window" title="Example 2.4.1"
        xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" >

<control="some-text" label value="Enter some text" />
<textbox id="some-text" />
<control="some-password" label value="Enter a password" />
<textbox id="some-password" maxlength="8" />

</window>
```

FIG. 6.11 – *Code d'un exemple de fichier XUL [Andersen et Deakin, 2004]*

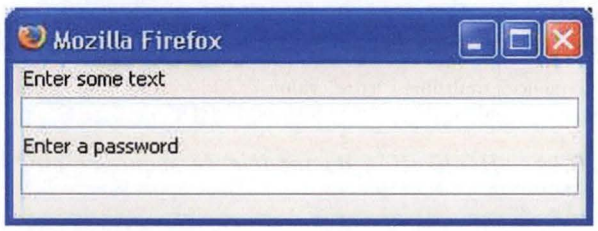


FIG. 6.12 – *Apparence de cet exemple de fichier XUL [Andersen et Deakin, 2004]*

Ces éléments de présentation peuvent générer des événements. Pour ce faire, le programmeur doit signaler dans la balise de l'élément l'événement qu'il veut écouter comme un clic, un passage sur l'élément, etc. En plus de l'événement, le programmeur donne aussi le code à exécuter ou le nom de la méthode qui réagira à l'événement. Cette méthode sera stockée dans un fichier JavaScript annexe.

Par exemple, la fenêtre 6.17 doit fermer lorsqu'on rempli le second champ. La figure 6.13 représente le code modifié (modification en rouge).

```
<?xml version="1.0" ?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css" ?>

<window id="example-window" title="Example 2.4.1"
        xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" >

<control="some-text" label value="Enter some text" />
<textbox id="some-text" />
<control="some-password" label value="Enter a password" />
<textbox id="some-password" maxlength="8" onkeypress[7]="window.close();" />

</window>
```

FIG. 6.13 – *Code d'un fichier XUL traitant un événement généré par la présentation [Andersen et Deakin, 2004]*

7. `onkeypress` correspond à l'événement : l'utilisateur remplit le champ de texte.

6.3.2 En pratique

Le modèle comprend des éléments d'IHM qu'il faudra traduire dans la syntaxe XUL. Or, le modèle connaît l'emplacement du contenu, mais pas la valeur de ce contenu. Pour mettre en évidence ces emplacements, nous allons présenter ces valeurs entre des étiquettes *add*.

La représentation du modèle de séquence (voir figure 6.3) sera la figure 6.14. Les informations à ajouter sont en rouge sur le schéma.

```
<script src=<add>statechart </add> />

<hbox>
<label value=<add>parametre.nom </add>":" />
<textbox id="nom" value=<add>valeur.nom </add> onkeypress="nom_complete();" />
<button id="parcourir-bouton" label="PARCOURIR" default="true"
                    oncommand="ouvrir_fichier();" />
</hbox>
<textbox id="sequence" multiline="true" value=<add>valeur.sequence </add>
                    onkeypress="sequence_complete();" />
```

FIG. 6.14 – *Réalisation du modèle de séquence en "XUL"*

Ensuite, nous allons réaliser le fichier JavaScript. Il se présente comme une liste de méthodes (ou procédures). Les noms de ces méthodes ont été définis dans le modèle pour réaliser le traitement d'un événement. Il suffit donc d'écrire le code correspondant aux transitions (provenant de la machine à états) avec la bonne méthode.

La figure 6.15 représente une partie du fichier JavaScript permettant l'exécution de la machine à états (voir figure 6.8). Le nom des méthodes (en mauve) fait référence aux éléments présentés en mauve du modèle (voir figure 6.14).

```
function nom_complete()
{
if (ETAT == "valeur vierge")
        {
        // nom = get_text()
            var textbox = docu-
ment.getElementById('nom');
        valeur.nom = textbox.valeur;
        ETAT = "nom complété";
        };
if (ETAT == "séquence complété")
        {
        // nom = get_text()
            var textbox = docu-
ment.getElementById('nom');
        valeur.nom = textbox.valeur;
        ETAT = "valeur complétée";
        // signaler au service
        }
}
```

```
function ouvrir_fichier()
{
if (ETAT == "valeur vierge")
        {
        // ouvrir fenêtre (ouvrir fichier)
        ...
        ETAT = "ouvrir fichier";
        };

function sequence_complete()
if (ETAT == "valeur vierge")
        {
        // séquence = get_text()
        ...
        ETAT = "séquence complété";
        // signaler au service
        }
...
}
...
```

FIG. 6.15 – *Réalisation du fichier JavaScript décrivant le dialogue de la figure 6.8*

Sur base du contenu, le modèle peut être complété afin d'obtenir un visualisateur. Pour

obtenir un fichier XUL, il faudra également ajouter la référence vers le fichier JavaScript gérant la conversation.

Pour construire un visualisateur XUL, nous avons associé un modèle XUL (voir figure 6.14) à un contenu (nom du paramètre : "séquence 1", valeur = " ") ainsi qu'une référence au fichier JavaScript (figure 6.15). Nous obtenons ainsi un fichier XUL : figure 6.16. Celui-ci pourrait être utilisé entre autre pour représenter le premier paramètre du service *algorithme Smith-Waterman* (voir figure 6.7 page 77).

```
<script src="sequence2-1.js"/>

<hbox>
<label value="sequence 1 :" />
<textbox id="nom" value="" onkeypress="nom_complete();" />
<button id="parcourir-bouton" label="PARCOURIR" default="true" oncommand="ouvrir_fichier();" />
</hbox>
<textbox id="sequence" multiline="true" value=""
                    onkeypress="sequence_complete();" />
```

FIG. 6.16 – *Réalisation d'un visualisateur en XUL du paramètre* séquence 1

Ce fichier XUL (voir figure 6.16) sera présenté à l'utilisateur selon la figure 6.17.

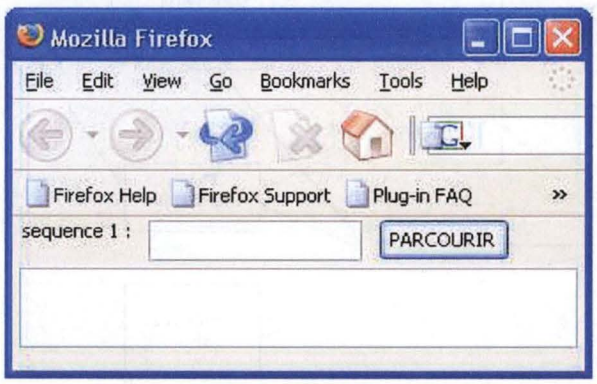


FIG. 6.17 – *Aspect de ce visualisateur du paramètre* séquence 1

6.4 Liens avec le reste de notre modélisation

Le chapitre 3 a permis d'identifier le besoin d'avoir de multiples représentations d'un même service pour présenter les différents niveaux de connaissance mais également pour s'adapter aux différentes visions propagées aujourd'hui par les fournisseurs. Nous avions également proposé un identifiant à l'entité `description d'IHM`. Nous allons donc maintenant réaliser le lien entre notre modélisation d'IHM et cet identifiant.

Une description d'IHM présente un contenu. Celui-ci provient d'une interface utilisateur. Elle prend donc en compte le niveau de l'utilisateur (débutant, normal, expert) par la signature qu'elle représente. De plus, tous les attributs de la signature doivent être présentés

à l'utilisateur.

Les interfaces utilisées par un même utilisateur doivent être représentées selon une certaine métaphore. En effet, le même type d'informations doit être représenté de manière relativement similaire[8]. Autrement dit, un type de donnée ne peut être visualisé qu'à l'aide d'un (voire éventuellement plusieurs) visualisateur(s). L'attribut métaphore (dans l'entité `description d'IHM`) permet de faire le lien entre toutes les interfaces d'un même utilisateur.

Chaque fournisseur créant des interfaces doit donc se baser sur une métaphore. Celle-ci fournit des modèles à utiliser. Il fournit également des comportements généraux des interfaces au moyen des machines à états.

Pour conclure, la figure 6.18 met en évidence les différentes étapes qui réalisent l'interface avec laquelle l'utilisateur dialoguera. Tout d'abord, la `description d'IHM` est identifiée par deux éléments (en vert sur le schéma) : `métaphore` (attribut) et `interface utilisateur` (entité). Ensuite, notre modélisation (en noir sur le schéma) permet de réaliser des fichiers XUL qui seront présentés sous forme d'interface à l'utilisateur (en bleu sur le schéma).

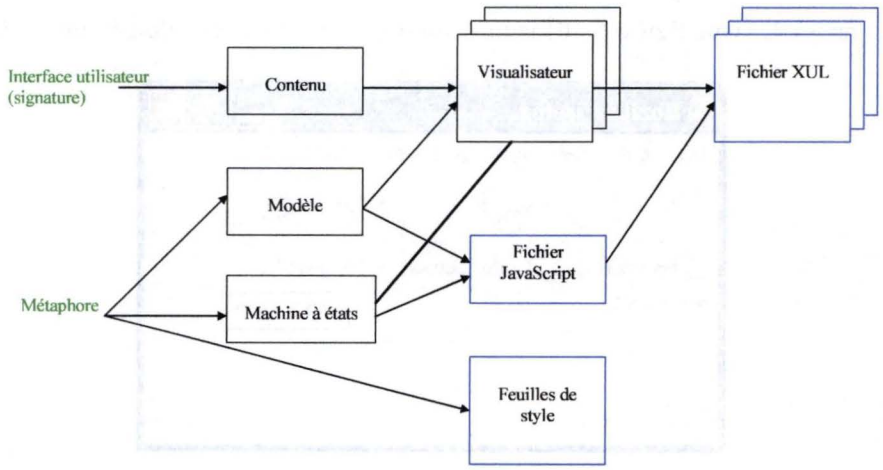


FIG. 6.18 – ***Les étapes de réalisation*** *Un modèle, une conversation se basent sur une apparence. Le contenu lui se base sur le niveau de l'utilisateur. Ces éléments permettent de produire les fichiers XUL et JavaScript.*

6.5 Conclusion

Dans ce chapitre, nous avons analysé la description d'IHM. Celle-ci est séparée en deux composants : la présentation et la conversation. D'une part, la présentation est décrite au moyen d'un visualisateur. D'autre part, la conversation est décrite au moyen d'une machine à états. Par exemple, le service *algorithme Smith-Waterman* est décrit par les éléments

8. Présenter la même information de manière similaire permet à l'utilisateur de mémoriser plus rapidement le lien existant entre l'apparence et l'information représentée.

suivants : le visualisateur du service (figure 6.7), le visualisateur de l'aide et la machine à états (figure 6.10).

Ensuite, nous avons présenté une technique XUL qui permet de mettre en pratique notre modélisation. Enfin, nous avons mis en évidence les éléments qui permettent de trouver l'Interface Homme Machine la plus adaptée pour chaque utilisateur.

Chapitre 7

Modélisation du protocole

Aujourd'hui, peu d'outils permettent à l'utilisateur moyen de réaliser aisément un enchaînement de services. Il doit donc réaliser individuellement chaque service et au besoin, attendre d'avoir certains résultats pour pouvoir exécuter le service suivant, sans compter les difficultés déjà présentées : les nombreux formats et les nombreux sites.

Dans ce chapitre, nous allons d'abord analyser les outils offerts par l'informatique pour modéliser un enchaînement. Ensuite, nous étudierons les outils qu'utilisent le biologiste pour mémoriser cet enchaînement. Enfin, nous proposerons une modélisation permettant à tous les utilisateurs de réaliser des enchaînements.

7.1 Outil de modélisation : les réseaux de Petri

La modélisation de processus au moyen des réseaux de Petri[van der Aalst et van Hee, 2002] s'applique en toute généralité [1].

Lors de la réalisation d'une application informatique, on appliquait la règle : "D'abord organiser, puis automatiser". Cela impliquait que le processus métier était géré par des personnes. Ensuite, une structure organisationnelle était développée où les groupes, les individus étaient chargés de la réalisation d'une tâche. Enfin, on examinait les tâches qui pouvait être réalisées par l'ordinateur ou plutôt par le système d'information. Or, cette approche ne prend pas suffisamment en compte les avantages d'un système d'information.

Une autre approche consiste à définir d'abord le processus de manière plus abstraite, sans considérer l'implémentation. Ensuite, on définit le système d'information ainsi que la structure organisationnelle. En fait, on décide si une tâche doit être réalisée par une personne ou un ordinateur. Le système d'information est ainsi plus complet et offre plus de possibilités. Faire usage pleinement du système d'information pourrait révolutionner les processus métier.

Dans cette section, nous allons d'abord définir le processus. Ensuite, nous allons montrer la formalisation de ces processus au moyen des réseaux de Petri. Ceux-ci sont en effet un bon outil pour modéliser et analyser les processus.

1. Cette modélisation n'est pas spécifique à la bioinformatique.

7.1.1 Définition d'un processus

Chaque entreprise définit son métier par des processus. Chaque processus comprend un ensemble de tâches à accomplir ainsi qu'un ensemble de conditions qui définissent l'ordre des tâches.

Une tâche est une unité logique de travail qui est réalisée par une ressource (personne, machine, etc.). Par exemple, le processus : traitement des constats d'assurance est réalisé par les tâches suivantes :

1. enregistrement du constat
2. paiement du montant
3. envoi d'une lettre expliquant le refus de paiement.

La ressource nécessaire dépend de la tâche. En effet, certaines tâches peuvent être réalisées par ordinateur sans interface humaine. D'autres tâches requièrent l'intelligence humaine pour un jugement, une décision.

Nous pouvons identifier quatre mécanismes qui structurent un processus en définissant l'ordre des tâches :

- *séquence* : des tâches réalisées dans un certain ordre;
- *sélection* : choix parmi plusieurs tâches;
- *parallélisation* : des tâches exécutées en parallèle;
  Les tâches suivantes doivent pouvoir attendre les exécutions de toutes ces tâches. Cela est appelé la *synchronisation.*
- *itération* : une tâche qui doit être répétée comme la révision d'une machine.

Tous ces mécanismes sont courants et permettent de modéliser tous les processus.

Pour exécuter un processus, une série de tâches doit être exécutée. La tâche automatique est réalisée par un programme. La tâche manuelle est réalisée par une personne compétente. Cette personne doit être mise au courant du travail à faire et peut prendre l'initiative de sa réalisation.

Il faut donc intégrer dans le système gérant le workflow des informations sur la structure organisationnelle pour connaître les compétences des différents employés ainsi que leur disponibilité. Le système sera ainsi capable de gérer l'assignation des tâches manuelles afin qu'elles s'exécutent le plus rapidement.

7.1.2 Modélisation des processus au moyen des réseaux de Petri

Un réseau de Petri est constitué de places et de transitions. Une place et une transition sont représentées respectivement par un cercle et un rectangle. Les places et les transitions sont reliées par des arcs (de place vers transition ou de transition vers place).

Un processus sera représenté par un réseau de Petri avec une entrée et une sortie. Les conditions sont représentées par des places et les tâches par des transitions.

La figure 7.1 représente un réseau de Petri modélisant le processus de traitement des constats d'assurance (présenté dans la section précédente). Le constat est enregistré puis soit un paiement est réalisé soit une lettre expliquant le rejet est envoyée.

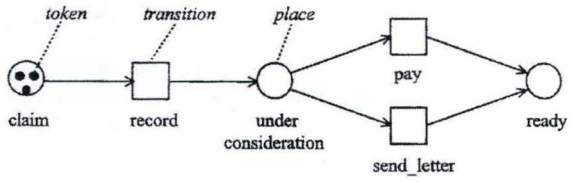


FIG. 7.1 – *Réseau de Petri modélisant un processus d'assurance [van der Aalst et van Hee, 2002]*

Une place peut contenir des jetons. La structure du réseau de Petri est fixe. Par contre la répartition des jetons entre les différentes places peut varier.

La transition *record* peut prendre des jetons de *claim* pour les placer dans *under consideration.* Cette action correspond à la réalisation de la transition. La transition ne peut être réalisée que lorsque toutes ces places en entrée contiennent un jeton. Dans la figure 7.1, seul la transition *record* peut être réalisée.

Nous allons maintenant expliciter les modélisations des différents mécanismes structurant les processus. Ainsi, la figure 7.2 représente une exécution séquentielle. La tâche 2 ne peut être exécutée que lorsque la tâche 1 sera terminée.

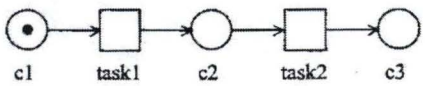


FIG. 7.2 – *Exécution séquentielle [van der Aalst et van Hee, 2002]*

Pour l'exécution en parallèle, les deux tâches peuvent être réalisées simultanément ou l'une après l'autre. La figure 7.3 illustre la modélisation de cette structure. Pour permettre ce type d'exécution alors qu'il n'y a qu'un jeton dans la place entrante, nous commençons par une tâche appelée AND-split : t1. A partir, d'un jeton en $c1$, t1 produit deux jetons : un en $c2$, un en $c3$. Comme la condition $c2$ est vérifiée, la tâche 1 peut être exécutée, de même que la tâche 2.

Enfin, t2 permet la synchronisation des deux tâches. En effet, la transition t2 ne peut être exécutée que lorsque les conditions $c4$ et $c5$ sont vérifiées.

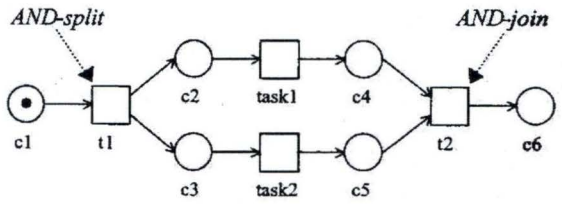


FIG. 7.3 – *Exécution en parallèle [van der Aalst et van Hee, 2002]*

Un choix doit pouvoir être fait entre différentes tâches, par exemple, une tâche ne doit être réalisée que dans les cas d'urgence, etc.

La figure 7.4 représente ce type d'exécution. Pour choisir la tâche à réaliser, ce réseau s'appuie sur les transitions $t11$ et $t12$ qui vérifient les pré conditions. En effet dès que la condition $c1$ est satisfaite, soit $t11$ soit $t12$ se déclenche ce qui entraînera l'exécution de la tâche 1 ou de la tâche 2. Dès qu'une des tâches est terminée, le OR-join s'assure qu'un jeton apparaisse en $c6$.

La figure 7.5 permet également de modéliser cette situation mais le processus $t1$ représente les règles de décisions. Cette figure présente mieux le choix entre les alternatives qui est réalisé sur base des valeurs du jeton. La transition $t1$ produit un jeton soit en $c2$, soit en $c3$ (mais pas un dans chacun).

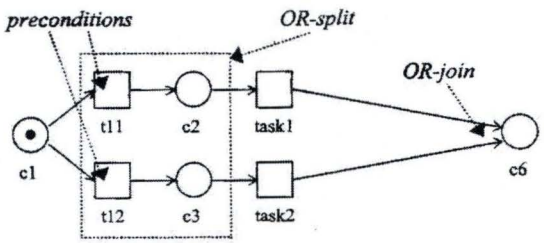


FIG. 7.4 – *Exécution sélective (1) [van der Aalst et van Hee, 2002]*

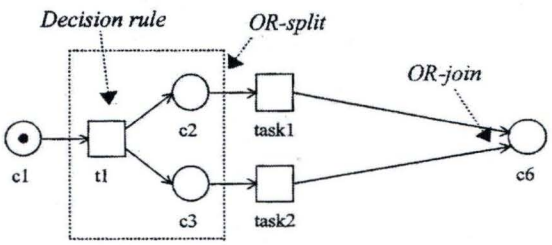


FIG. 7.5 – *Exécution sélective (2) [van der Aalst et van Hee, 2002]*

La structuration : répétition se base sur une tâche et une condition à remplir. Deux types

existent : le *repeat...until* (voir figure 7.6) et le *while...do* (voir figure 7.7). Dans le premier cas, la tâche doit être réalisée avant de tester la condition. Dans le second cas, la condition est testée avant de réaliser la tâche.

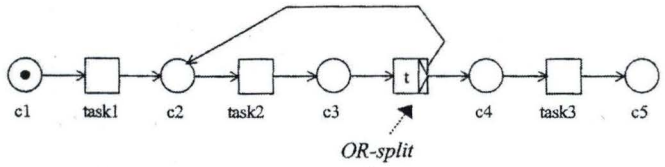


FIG. 7.6 – *Exécution itérative (1) [van der Aalst et van Hee, 2002]*

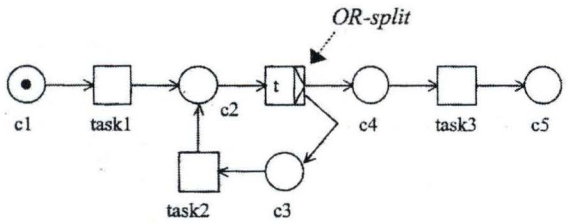


FIG. 7.7 – *Exécution itérative (2) [van der Aalst et van Hee, 2002]*

Nous avons considéré jusqu'à maintenant que la présence d'un jeton dans les places entrantes provoquait instantanément la réalisation de la transition mais d'autres événements extérieurs influencent cette réalisation : par exemple, une tâche manuelle ne commence que lorsque l'employé commence le traitement qu'il doit réaliser.

La figure 7.8 montre la représentation graphique de ces événements extérieurs. Les tâches 2 et 4 sont réalisées par des humains (il faut donc que les personnes puissent prendre en charge la réalisation de la transition). La tâche 3 dépend du temps (la transition doit être réalisée à une certaine heure) et la tâche 1 attend un événement extérieur (provenant d'une application,..). Seule la tâche 5 est automatique.

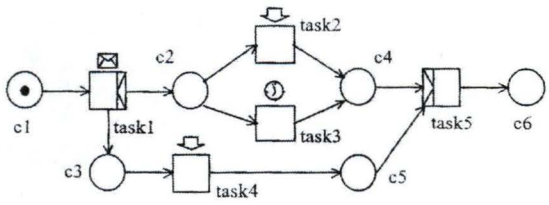


FIG. 7.8 – *Plusieurs formes de transitions [van der Aalst et van Hee, 2002]*

7.2 Les techniques de modélisation d'un protocole

Un protocole en biologie décrit précisément les conditions, le déroulement d'une expérience. Pour réaliser une expérience, le biologiste doit réaliser des tâches, nous pouvons donc considérer un protocole comme un processus.

Comme la bioinformatique permet également de réaliser des expérience, les biologistes modélisent les protocoles de bioinformatique comme les protocoles de biologie. Cette modélisation se base sur les tâches que le biologiste doit exécuter.

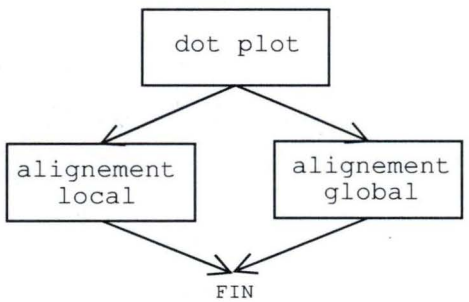
A cause de la difficulté pour le biologiste de réaliser des enchaînements de services[2], plusieurs projets tentent de proposer des outils pour aider à cette réalisation. On note que l'apparition dans le domaine des systèmes distribués et les recherches déjà effectuées contribuent fortement à la réalisation de ces outils. Pour pouvoir exécuter un enchaînement, le biologiste le modélise en définissant les programmes utilisés ainsi que les liens les unissant.

Dans cette partie, nous allons étudier la réalisation d'un protocole au moyen de ces deux modélisations. Ensuite, nous proposerons une approche mixte qui nous permettra enfin d'intégrer ces services "protocole" dans notre catalogue de service.

7.2.1 Approche orientée tâche

Un protocole est d'abord représenté par un schéma mettant en avant les liens entre les étapes. La sémantique du schéma est semblable à celle d'un réseau de Petri mais seules les transitions sont représentées. De plus, les mécanismes de structuration sont peu utilisés car le schéma est commenté.

La figure 7.9 représente un protocole. Celui-ci explicite les bonnes pratiques pour réaliser un alignement de deux séquences. Le biologiste doit d'abord réaliser un dot plot[3]. Ensuite, d'après le graphique résultat, il est capable de décider si les séquences sont proches ou éloignées et donc choisir le type d'alignement le plus adapté.



Etape 1 : dot plot
Cette étape permet d'identifier les régions de similarité de deux séquences

Etape 2 : alignement local OU alignement global
D'après l'étendue de la zone de similarité, aligner les séquences selon une des méthodes proposées.

FIG. 7.9 – *Protocole de bioinformatique pour réaliser un* alignement pairwise

Ensuite, chaque étape détaille la tâche par une liste d'instructions ainsi que d'éventuels

2. La difficulté provient des nombreux formats, de la nécessité de faire interagir des sites distants.
3. Le dot plot est un graphique mettant en évidence les régions de similarité de deux séquences.

conseils. Les instructions comprennent le nom du programme à utiliser avec éventuellement l'adresse web, les valeur des paramètres, etc. Les conseils sont principalement utilisé dans les tutoriaux (comme 2Can[EBI, 2004a] et ember[UMBER *et al.*, 2004]) pour appréhender le fonctionnement du service et aider à l'interprétation du résultat.

Par exemple, la figure 7.10 représente l'étape 1 du protocole (voir figure 7.9).

ETAPE 1 : réalisation d'un dot plot.

Pour visualiser les zones de similarité, nous allons réaliser un dot plot.

i) Visitez la page de *dotmatcher* sur le site de l'EBI et remplissez les valeurs des deux premiers champs avec vos séquences.
ii) Exécutez *dotmatcher* et examinez les résultats.
iii) L'information recherchée n'apparaît pas clairement sur le graphique? Changez le paramètre *window* et adaptez le *threshold*, puis ré-exécutez le programme. (diminuer la taille de la fenêtre amène plus d'information mais aussi plus de bruit)

Réflexions...

- Quels est l'impact du paramètre *window*?
- Et l'impact du paramètre *threshold*?

FIG. 7.10 – *Etape 1 du protocole 7.9*

Le fait que ce protocole soit utilisé par le biologiste pour réaliser une expérience, et qu'il corresponde à une modélisation informelle du travail effectué, induit que ces protocoles contiennent une masse d'informations sur le travail des biologistes. De plus, on peut comparer ce protocole à un workflow car il permet également de standardiser une tâche.

Evaluation par rapport aux réseaux de Petri. La modélisation de ces protocoles est informelle, néanmoins l'information présentée est utilisable par le biologiste pour exécuter le workflow. Mais cette information n'est pas utilisable par l'ordinateur.

Cette modélisation permet d'exprimer les actions que l'utilisateur doit réaliser. En général, il devra choisir entre plusieurs alternatives ou encore modifier une donnée. Dans cet exemple, l'utilisateur doit réaliser deux actions : juger la qualité d'un résultat et choisir le programme le plus adapté.

Comme cette modélisation ne représente que le travail d'un seul homme, le mécanisme de structuration "parallélisation" n'existe pas. Lorsque le protocole suggère d'utiliser deux programmes en parallèle, il propose à l'utilisateur de réaliser les tâches suivantes de manière séquentielle:

1. remplir le formulaire du premier programme puis lancer son exécution.
2. remplir le formulaire du second programme puis lancer son exécution.

3. attendre les deux résultats.

7.2.2 Approche orientée programme

Nous avions déjà souligné la difficulté d'enchaîner des services. Il était donc indispensable de disposer d'un outil permettant de réaliser un enchaînement de services. A l'heure actuelle, quelques projets tentent de réaliser un éditeur de workflow bioinformatique, gérant également l'exécution de ce workflow comme Taverna[EBI *et al.*, 2004].

Pour réaliser un workflow, l'utilisateur doit réaliser un dessin représentant les programmes utilisés ainsi que les liens entre eux. Taverna présentent les programmes comme des rectangles jaunes ; le lien entre les programmes est réalisé par les données (représentées par des triangles bleus) : une flèche partant de la donnée vers le programme signifie que la donnée est un paramètre en entrée du programme, une flèche partant du service vers la donnée signifie que la donnée est un paramètre en sortie.

Dans cette modélisation, notre protocole (voir figure 7.9) n'est pas réalisable car on ne sait pas modéliser les choix réalisés par l'utilisateur lors de l'exécution. Le choix doit donc être réalisé a priori lors de la modélisation du protocole. Nous faisons donc l'hypothèse que les séquences sont éloignées : le protocole se réduit donc à un dot plot et un alignement local (qui peuvent désormais être réalisés en parallèle). La figure 7.11 est la représentation de ce protocole.

Le service *dot plot* (programme *dotmatcher*) prend en entrée les deux séquences qu'il doit comparer tout comme le service *alignement local* (programme *water*). Ces deux programmes produisent tous deux un fichier.

La modélisation utilisée dans Taverna présente quelques similitudes avec les réseaux de Petri. On pourrait comparer un rectangle programme avec une transition et, un triangle donnée avec une place. Néanmoins, la sémantique des réseaux de Petri est différente : une place représente une condition et non une donnée utilisée comme paramètre.

Tout d'abord, un triangle donnée permet de réaliser tous les rectangles programmes auxquels il est lié par des flèches input en dupliquant cette donnée; contrairement aux réseaux de Petri, où un jeton ne peut être dédoublé que par une transition spécifique.

De plus, un triangle donnée ne peut provenir que d'un rectangle transition. En effet, une donnée ne peut avoir qu'une provenance, on ne peut imaginer une donnée créée par plusieurs programmes.

Enfin, contrairement au réseau de Petri, cette modélisation possède plusieurs triangles données en entrée et en sortie. Ces données correspondent à la signature du protocole. Par exemple, la signature du protocole de la figure 7.11 est représenté par :

$$f : sequence, sequence \longrightarrow graphique, alignement$$

Tout comme le service, le protocole ne peut être exécuté que lorsque toutes les données en entrée ont des valeurs et se termine lorsque les données en sortie ont été produites.

Cette modélisation peut être considérée comme une adaptation des réseaux de Petri. Il faut ajouter au schéma certaines contraintes (une place ne peut provenir que d'une seule transition), enlever certaines contraintes (un processus a une place entrante et une place sortante).

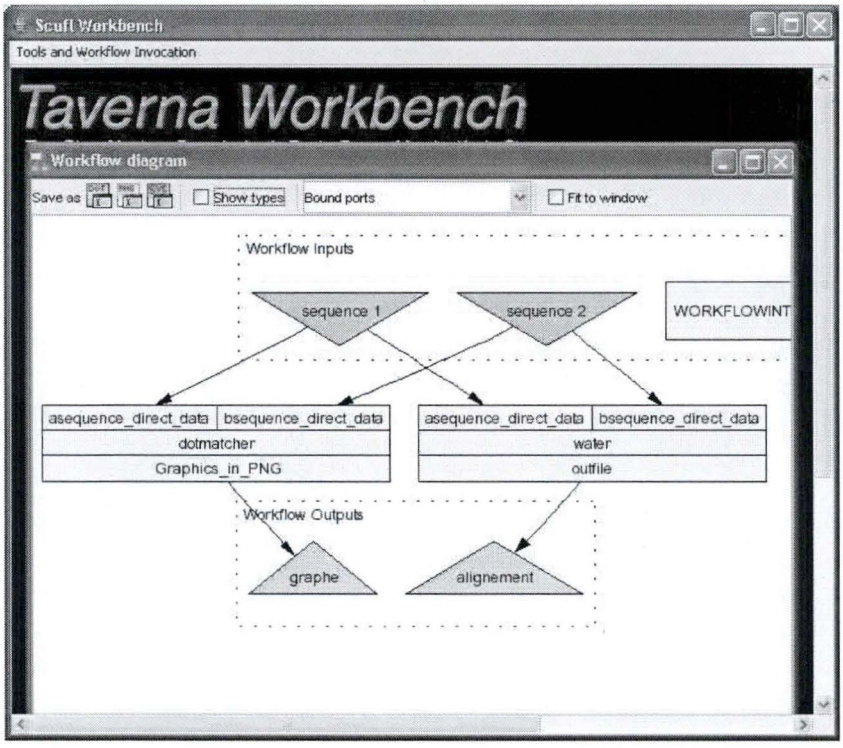


FIG. 7.11 – *Réalisation au moyen de Taverna du protocole* alignement pairwise *[EBI et al., 2004]*

Enfin, une place donnée doit contenir un jeton dès qu'une transition est réalisable dans une nouvelle configuration autrement dit dès que la signature du service a changé. Par exemple, la place *sequence 1* contient 2 jetons : un à destination de la transition *dotmatcher*, un autre à destination de la transition *water*

A cause de toute ses adaptations, cette modélisation est plus pratique que les réseaux de Petri pour présenter les programmes bioinformatiques et surtout ce qui les lie : les données.

Evaluation par rapport aux réseaux de Petri. La modélisation utilisée par Taverna est formelle. La représentation d'un protocole par l'utilisateur permet à Taverna de pouvoir exécuter cet enchaînement. Il manque néanmoins une possibilité de commenter le protocole et ainsi présenter ce protocole également comme une expérience et non simplement comme un enchaînement de services.

Cette modélisation ne représente que le travail automatique et ne permet pas de représenter les tâches manuelles : les choix parmi plusieurs alternatives, une validation de données, le remplissage d'un formulaire, etc.

Les mécanismes de structuration "sélection et itération" ne sont pas réalisables dans cette modélisation. En effet, on peut seulement postposer la réalisation de la transition à la fin de l'exécution d'un autre programme. Il n'y a pas d'autres moyens de soumettre le déclenchement d'un programme à une condition quelconque : une expression logique entre des paramètres ou un choix réalisé par l'utilisateur. Il n'est donc pas possible de réaliser des

parcours différents d'après les valeurs des données.

7.2.3 Approche mixte

Ces deux approches sont diamétralement opposées. L'une est à destination de l'humain, l'autre est à destination de l'ordinateur. Nous voudrions proposer une approche permettant à la fois l'exécution automatique et l'intégration de tâches manuelles (comme la validation, le choix) décrites dans l'approche tâche.

L'approche programme nous semble la plus intéressante au niveau exécution tandis que l'approche orientée tâche nous apparaît comme la plus complète. Or les réseaux de Petri permettent de modéliser non seulement les tâches automatiques mais également les tâches manuelles.

Nous allons donc modéliser le protocole orienté tâche au moyen des réseaux de Petri mais également au moyen de l'approche programme pour les services automatiques.

Tout d'abord, nous allons définir les différentes tâches citées dans la description du protocole. Cette première analyse identifiera les points importants de la description à la fois les services du catalogue mais aussi les tâches humaines.

Par exemple, la figure 7.12 représente les tâches nécessaires pour réaliser chaque étapes du protocole 7.9. Les tâches *dot plot, alignement local* et *alignement global* sont évidentes, elles correspondent aux appels de service. La tâche *satisfait ?* permettra à l'utilisateur de visualiser la donnée et de donner son niveau de satisfaction. La tâche *choix* permettra à l'utilisateur de choisir entre un alignement local ou un alignement global.

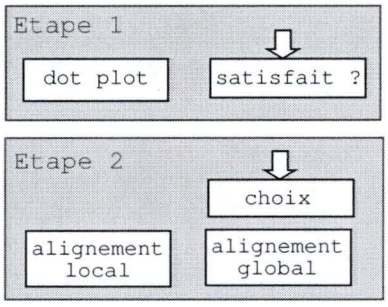


FIG. 7.12 – *Vision statique du protocole* alignement pairwise

Ensuite, chaque service doit être défini par une signature. Cette signature permettra de mettre en évidence la production/transformation de donnée de chaque tâche. Les services automatiques sont déjà définis dans le catalogue de services par les signatures[4] suivantes:

$$dot\ plot : sequence,sequence \longrightarrow graphique$$

$$alignement\ local : sequence,sequence \longrightarrow alignement$$

$$alignement\ global : sequence,sequence \longrightarrow alignement$$

4. Ces signatures sont simplifiées, elles ne contiennent que les parties input et output, pas les autres paramètres.

Les services manuels ne sont pas définis. Cette signature permettra de souligner les indices donnés à l'utilisateur (input) et les données, décisions, etc. que l'utilisateur doit produire (output). De plus, dans le cadre d'un choix, il est également important de définir les alternatives.

Par exemple, le service *satisfait ?* est défini par

$$satisfait? : graphique \longrightarrow \{YES,NO\}$$

$$satisfait?(x) := \textbf{if } x \textit{ présente information } \textbf{then } YES \textbf{ else } NO$$

et le service *choix* est défini par

$$choix : graphique \longrightarrow \{1,2\}$$

$$choix(x) := \textbf{if } x \textit{ présente ligne continu } \textbf{then } 2(\textit{alignement global}) \textbf{ else } 1(\textit{alignement local})$$

Enfin, il faut présenter les liens entre ces différentes signatures. Les services automatiques sont présentés selon l'approche programme (les ronds verts correspondent à des données) tandis que le reste de la modélisation s'appuie sur les réseaux de Petri classiques.

Par exemple, la figure 7.13 représente l'enchaînement des tâches nécessaires pour réaliser chaque étapes du protocole 7.9.

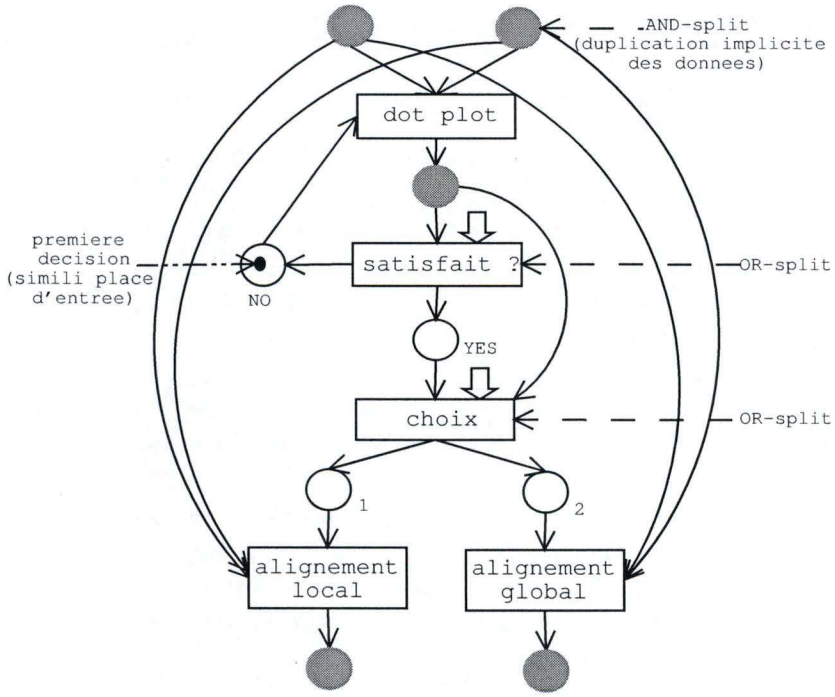


FIG. 7.13 – *Vision dynamique du protocole* alignement pairwise

Les tâches de décisions vont générer le jeton qui influencera le choix de la transition réalisable. Donc, il n'y a pas de place d'entrée contenant le jeton de départ (le départ ne peut se faire que lorsque toutes les données d'input sont complétées). Cela ne pose pas de problème sauf si le premier mécanisme de structuration est de type *repeat...until*. En effet, le jeton doit être présent avant de pouvoir réaliser la première étape. Par exemple, notre protocole commence par une telle transition : *repeat dot plot until satisfait ?*. Une place a donc dû être considérée comme simili-entrée. Le départ ne peut se faire que lorsque les données d'entrée sont complétées et qu'un jeton se trouve dans la place de simili-entrée. Il faudra donc mettre un jeton dans cette place lors de la demande par l'utilisateur de l'exécution du protocole.

Evaluation par rapport aux réseaux de Petri. Notre modélisation est formelle et complète. L'information présentée est à destination de l'ordinateur mais elle intègre les éléments permettant d'interagir avec l'utilisateur. Néanmoins l'approche tâche sera toujours la bienvenue comme documentation pour le biologiste.

7.3 Liens vers le reste de la modélisation

Un protocole permet de documenter un service. Il fait donc référence à un service offert à l'utilisateur. Dans cette partie, nous allons tenter de relier les protocoles à notre catalogue de services. Jusqu'à présent, nous avions principalement considéré que les services présents dans notre catalogue était des services simples - c-à-d lier à un seul type de services. Nous n'avions pas pensé le service en tant qu'ensemble de services. Nous allons donc voir comment nous pouvons caractériser les protocoles en terme de service et de description d'IHM.

7.3.1 Liens vers le service

Le protocole par l'assemblage d'autres services permet la création de nouveaux services. Ces services seront abstraits, en effet, ils ne sont pas instanciables. Ces services seront réalisés par un ensemble d'instances de service (chaque instance correspondra à un service automatique du schéma).

Par exemple, le protocole défini dans ce chapitre (voir figure 7.13) documente le service abstrait *alignement pairwise*.

Ce nouveau service doit avoir des caractéristiques de service : la signature, une description fonctionnelle pour pouvoir en offrir l'accès à tous les utilisateurs.

La signature sera générée à partir des signatures des différentes tâches. En effet, les données nécessaires en entrée sont d'une part : les places sans provenance du schéma (dans notre exemple : séquence 1 et 2) et tous les autres paramètres des différents services automatiques appelés ainsi que les choix réalisés par les utilisateurs. Les données en sortie seront les résultats intermédiaires et finaux.

La signature du service sera donc composée de :

- $input = \bigcup_{x \in S} input_x \setminus output$
- $autres\ parametres = \bigcup_{x \in S} autres\ parametres_x + \bigcup_{y \in C} output_y$

- $output = \bigcup_{x \in S} output_x$

  où S et C correspond respectivement à l'ensemble des services automatiques et manuels utilisés.

Par exemple, notre protocole aura comme signature :

- *input* = sequence 1, sequence 2
- *autres parametres* = window size, threshold, satisfait, choix, gap opening penalty, gap extension penalty, matrice de substitution
- *output* = dot-plot, alignement local, alignement global.

La description fonctionnelle sera réalisée par l'utilisateur lors de l'enregistrement de son service. Par exemple, le protocole défini dans ce chapitre aura comme description fonctionnelle la table 7.1.

nom	**valeur**
nom	alignement pairwise
description	permet de comparer deux séquences
traitement métier	alignement pairwise
algorithme	néant

TAB. 7.1 – *Description fonctionnelle du protocole* alignement pairwise

7.3.2 Liens vers l'Interface Homme Machine

Ce nouveau service a également besoin d'une interface utilisateur. En effet, une fois ce nouveau service créé, l'utilisateur doit pouvoir l'invoquer et donc dialoguer avec ce service, particulièrement dans le cadre des tâches manuelles.

Comme la signature du protocole est construit sur base des signatures des services, la présentation du protocole s'appuiera sur les présentations des services.

La présentation des tâches automatiques est déjà stockée dans le catalogue de service. Par exemple, la figure 6.7 (page 77) représente le service alignement local.

Par contre, les tâches manuelles doivent être définies. Leur caractéristique majeure est de devoir interagir avec l'utilisateur. Pour réaliser cette présentation, nous allons nous baser sur leur signature. D'une part, l'input représente les indices donnés à l'utilisateur. L'interface doit donc permettre à l'utilisateur de visualiser ces données. D'autre part, l'output représente la donnée, la décision produite par l'utilisateur. L'interface doit donc donner à l'utilisateur les moyens de produire une donnée, une décision, etc.

Par exemple, la tâche manuelle *choix* a été représentée par la figure 7.14. Elle est constituée de deux parties : une représentant la donnée (input), l'autre permettant à l'utilisateur d'interagir et de renvoyer au système son choix (output). L'output du service sera 1 si l'utilisateur appuie sur le bouton alignement local, 2 s'il appuie sur le second bouton : alignement global.

Le nombre de services présents dans un protocole peut être élevé. Il nous semble donc important de prévoir des modèles pouvant contenir plusieurs services.

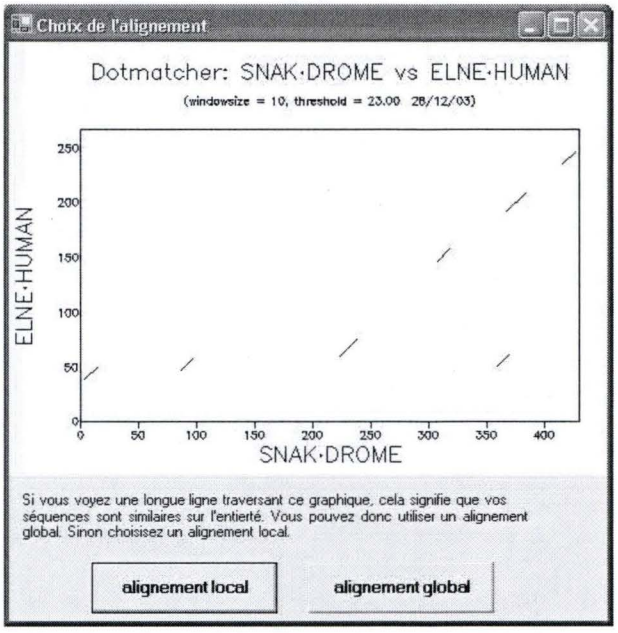


FIG. 7.14 – *Présentation de la tâche* choix

Par exemple, la figure 7.15 représente une interface qui prend en compte le service *satisfait ?* et le service *dot plot* prévu si l'utilisateur souhaite ré-exécuter ce service. En effet, les différents paramètres à modifier pour améliorer le résultat du dot-plot sont présentés.

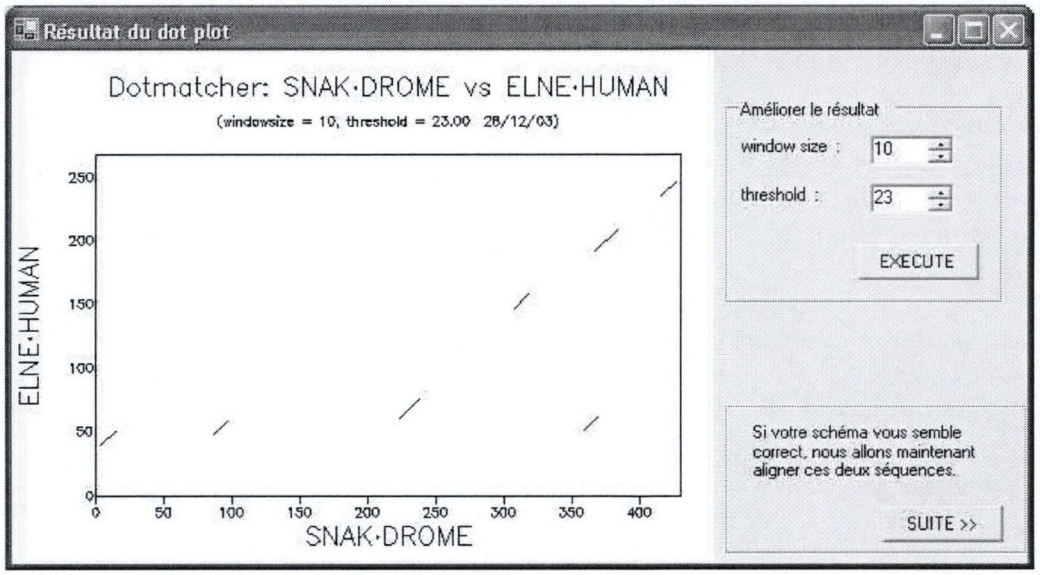


FIG. 7.15 – *Présentation de la tâche* satisfait ?

Le dialogue devra être très proche du protocole. En effet, il exprime également une notion d'enchaînement mais les bases sont différentes. Le protocole enchaîne les services, le dialogue enchaîne les interfaces permettant aux utilisateurs de paramétrer les différents services.

Dans le cas d'un service automatique, l'interface ainsi que la conversation du service sont déjà réalisées. Il suffit donc de lancer la conversation et d'attendre que cette conversation soit finie. La machine à états du service préviendra par un signal *exe* (en général, ce signal sera envoyé lorsque l'utilisateur appuie sur le bouton EXECUTER). La translation du protocole vers la machine à états est présentée à la figure 7.16.

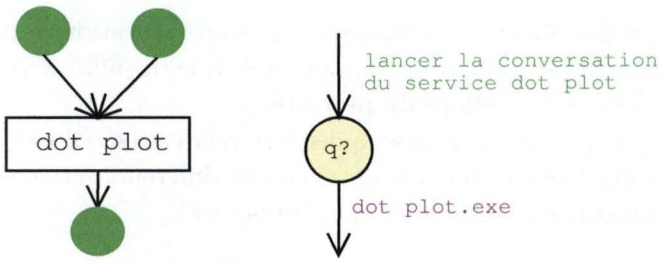


FIG. 7.16 – *Liens entre protocole et machine à états (service automatique)*

Dans le cas d'un service manuel, la présentation vient d'être définie. Il faut donc également définir leur conversation.

Cette présentation s'appuie sur des données à présenter à l'utilisateur. Donc, pour pouvoir présenter le service à l'utilisateur, les services réalisant ces données doivent être terminés. Il faut donc que lors de l'exécution du protocole, la conversation soit prévenue des réalisations des données pour ensuite pouvoir afficher cette donnée.

La présentation propose également à l'utilisateur les différentes alternatives. La décision de l'utilisateur parvient à la conversation au moyen d'événements. La conversation doit donc communiquer ce choix au protocole. Pour ce faire, une nouvelle action est donc disponible : *forward* qui envoie au protocole la décision de l'utilisateur [5].

La figure 7.17 représente le translation du protocole vers la machine à états pour le service choix (présentation : voir figure 7.14).

5. Lorsque l'utilisateur doit fournir une donnée au service, le dialogue devra récupérer la donnée dans la présentation pour la faire parvenir au protocole.

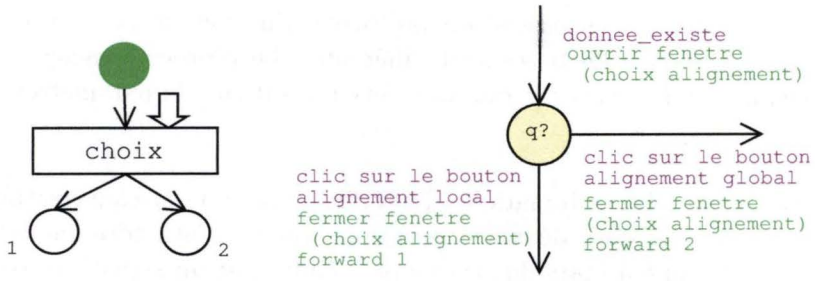


FIG. 7.17 – *Liens entre protocole et machine à états (service manuel)*

Il est ainsi possible de définir pour chaque service sa conversation. A partir de ces conversations, nous pouvons réaliser la conversation du protocole. Il suffit de baser l'enchaînement des services sur l'ordre des transitions du protocole.

Si une présentation permet de réaliser plusieurs services, sa conversation doit prendre en compte les différents événements et les actions des différents services (seules les actions de type ouvrir et fermer des fenêtres peuvent être omises).

Par exemple, la figure 7.18 présente la machine à états de notre protocole.

Cette machine à états anime les fenêtres : dot-plot, satisfait? et dot-plot (figure 7.15), choix (figure 7.14), alignement local (figure 6.7) et alignement global.

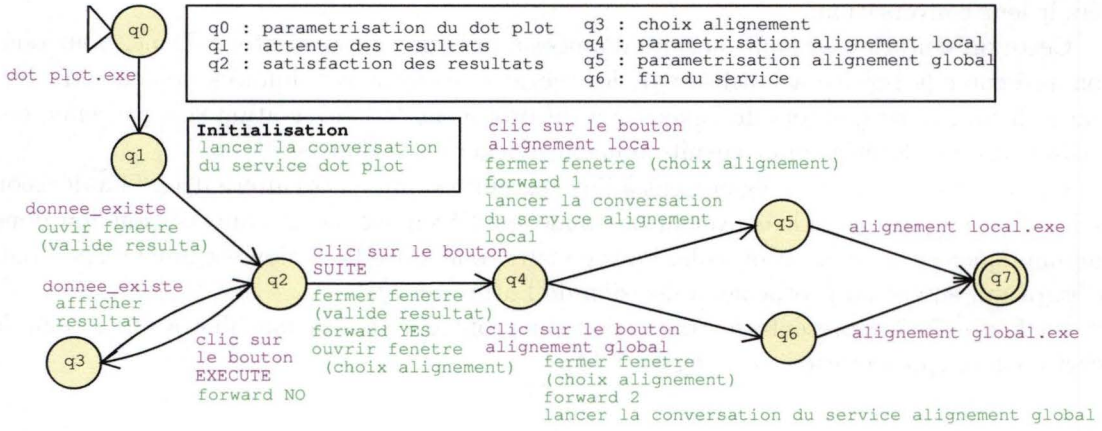


FIG. 7.18 – *Dialogue du protocole* alignement pairwise

7.4 Conclusions

Les précédents chapitres ont mis en évidence la difficulté de réaliser automatiquement un enchaînement de services. Jusqu'il y a peu la seule possibilité consistait à écrire un script (petit programme) généralement en Perl. Or ce besoin de collaboration devient de plus en plus important.

Dans ce chapitre, nous avons d'abord étudié les techniques de modélisation utilisées en informatique pour prendre en compte un enchaînement de tâches. Nous avons ensuite analysé les techniques utilisées aujourd'hui pour modéliser ces enchaînements appelés protocoles. Enfin, nous avons réalisé notre propre modélisation. Celle-ci permet d'enchaîner non seulement les programmes mais permet également à l'utilisateur d'interagir avec le service.

Enfin, un enchaînement de services sera considéré comme un service et sera donc décrit par une signature ainsi que par une IHM. Cette IHM ne présente pas la modélisation de l'enchaînement mais la paramétrisation des différents services utilisés, elle est donc à portée de tous les utilisateurs contrairement à la modélisation de l'enchaînement (aussi bien notre modélisation du protocole que Taverna) qui nécessite plus d'expérience.

L'annexe B propose la modélisation de notre exemple fil rouge.

Chapitre 8

Perspectives

Ce mémoire a permis de modéliser les principaux éléments constitutifs de la bioinformatique : la donnée, le service, l'IHM et enfin le protocole. Néanmoins, ce travail de modélisation peut être complété pour permettre une simplification accrue de l'usage de la bioinformatique.

Ainsi, nous avons défini une modélisation complète mais nous n'avons pas mis en évidence l'influence de cette modélisation sur la présentation d'une donnée.

Ensuite, la présentation d'un service dans notre modélisation n'est réalisée qu'au moyen d'un formulaire. Or ce formulaire n'est pas adapté à l'utilisateur débutant. Une alternative serait un assistant autrement dit une interface qui se renseigne sur les intentions de l'utilisateur au moyen de questions.

Dans ce chapitre, nous allons rapidement évoquer ces évolutions que nous souhaiterions voir ajouter à notre modélisation.

8.1 Visualisation de la donnée

Dans le chapitre 4, nous avons mis en évidence une modélisation complète de la donnée. Celle-ci comprend non seulement le contenu de la donnée mais également toutes les informations importantes pour le biologiste afin de faciliter sa comprehension de ces données.

Ces informations devraient être accessibles au moyen d'une présentation adéquate comme par exemple un menu permettant d'accéder à l'information sur les références identiques et similaires.

En particulier, la visualisation d'un résultat d'exécution doit être soignée pour mettre en évidence le passé, le présent et le futur de ce résultat.

Le passé est représenté par les paramètres ayant permis de réaliser la donnée. Ce passé doit pouvoir être modifiable, en effet, il est difficile pour le biologiste d'obtenir un résultat satisfaisant dès la première exécution. A partir de ses observations sur la donnée, l'utilisateur voudra peut-être ré-exécuter le même service mais avec des paramètres différents. De cette façon, il améliorera sa donnée.

Le présent sont les commentaires réalisés par le biologiste. Ces commentaires devraient normalement lui permettre d'évaluer a posteriori son expérience, sa donnée.

Le futur, enfin, présente les différents services intéressants à effectuer. Proposer directement ces services à l'utilisateur permettra de faciliter son accès à tous ces services car il ne

devra pas d'abord rechercher ce service et permettra également à l'utilisateur de découvrir des services qu'il ne connaît pas.

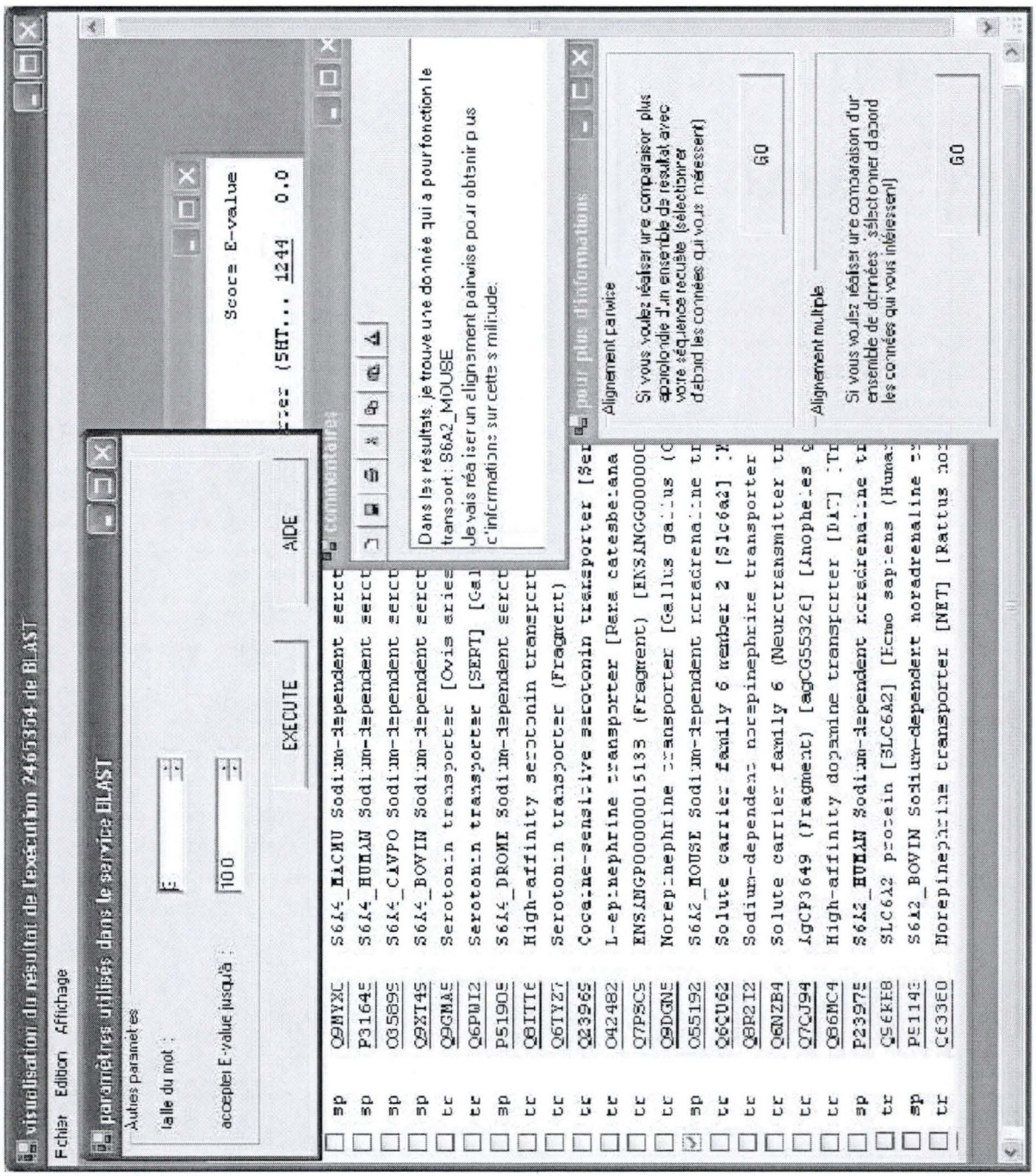


FIG. 8.1 – *Visualisation des résultats*

Par exemple, la figure 8.1 présente la visualisation d'un blast réalisé dans le cadre, par exemple, de l'expérience basique réalisée en début de mémoire. Ce même résultat est

aujourd'hui présenté dans la figure 1.7 (page 15).

Les paramètres ayant permis de réaliser cette donnée sont la taille du mot et le niveau maximum de l'E-value. Si le biologiste ne dispose pas d'assez d'entrées, il peut alors très facilement accéder à ces paramètres (augmenter le niveau maximum de l'E-value) et ainsi améliorer sa donnée (augmenter le nombre d'entrées présentées).

Le biologiste peut également faire des commentaires sur sa donnée. Ici, il indique avoir trouvé des éléments confirmant son hypothèse.

Enfin, le système propose au biologiste deux services. Ces services sont une suite logique dans diverses expériences. Dans notre expérience, l'utilisateur choisit l'alignement pairwise et poursuit ainsi cette expérience, sans recherche de ce service.

8.2 Autres présentations du service

La présentation du service est proposée sous la forme d'un formulaire mais cette représentation utilisée aujourd'hui ne permet pas aux utilisateurs débutants de bien appréhender le fonctionnement du service pour pouvoir le paramétrer correctement.

Une alternative au formulaire pourrait être un assistant autrement dit une interface qui pose des questions à l'utilisateur pour trouver le service ainsi que les paramètres ad-hoc. Comme par exemple, la figure 8.2 qui représente la première étape dans la réalisation d'un alignement.

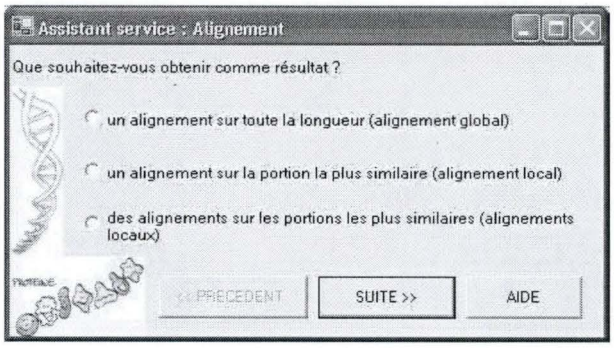


FIG. 8.2 – *Présentation de l'assistant Alignement*

Pour réaliser un tel assistant, il faut se baser sur les règles de choix d'un programme, par exemple voir la table 5.13 (page 64) et réaliser une table du même genre concernant le choix des paramètres Ensuite, il faut traduire ces tables dans une certaine modélisation qui permette à partir des questions de trouver le service idéal.

Notre première idée de modélisation est l'arbre de décision. par exemple, la figure 8.3 donne pour l'alignement les différentes règles (la figure 8.2 est la représentation du premier palier de décision.) Il faudrait vérifier néanmoins que cette modélisation est suffisamment complète pour permettre d'exprimer les différents services. Dans le cas contraire, il faudrait pouvoir exprimer ces contraintes au moyen d'un autre formalisme.

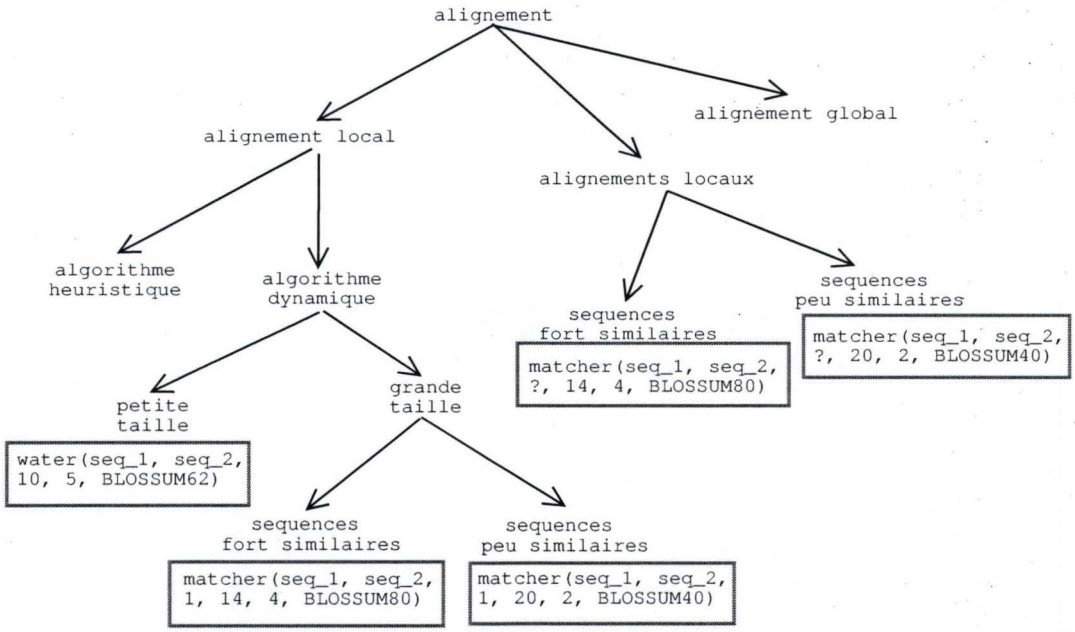


FIG. 8.3 – *Arbre de décision pour le service alignement*

Conclusion

La bioinformatique peut être vue aujourd'hui comme une collection de sites autonomes. Cette autonomie se traduit par des présentations différentes qui demandent à l'utilisateur un effort d'adaptation. De plus, la grande diversité des logiciels d'analyse, les nombreux formats de données sont également des freins à l'utilisation de la bioinformatique pour des utilisateurs non initiés.

Des mémoires précédents [Debaisieux et Desouza, 2003, Buyle et Dallons, 2003] ont permis la réalisation d'un prototype de systèmes distribués appliqué à la bioinformatique. Celui-ci permettait de gérer l'accès aux services et les formats, facilitant ainsi le travail du biologiste.

Cependant, ces travaux n'ont que très peu abordé la modélisation d'un service et des différents éléments dépendant de celui-ci comme l'IHM. Notre travail s'est inscrit dans la suite de ces projets. Son objectif était de modéliser le service et ce qui y avait trait : les données, les IHM et les enchaînements de services. Cette modélisation devra pouvoir être utilisée par les fournisseurs de services mais dans certains cas aussi par les biologistes.

Dans ce travail de modélisation, nous avons essayé de faciliter l'accès à la bioinformatique pour tous les utilisateurs.

Cette modélisation devait d'abord fournir à chaque utilisateur une vue homogène des services dont il avait besoin. La description du service est donc réalisée par le système et non plus par les différents fournisseurs.

Une uniformisation pour tous les utilisateurs de la caractérisation d'un service n'était pas souhaitable. En effet, les attentes de l'utilisateur sont différentes car leur niveau (débutant, normal, expert) et leurs habitudes en terme de présentation sont différents. Le service est décrit donc par plusieurs *interfaces utilisateurs* ainsi que par différentes interfaces homme machine.

Ensuite, pour aider l'utilisateur à gérer la diversité des logiciels, nous avons créé des *services abstraits*. Ceux-ci sont des nouveaux services qui représentent un type de service autrement dit une généralisation d'un ensemble de services.

L'utilisateur peut donc accéder à un ensemble de services au moyen de l'interface d'un seul service. Le système trouvera dans cet ensemble le service le plus adapté qui réalisera la demande de l'utilisateur.

En outre, la modélisation d'enchaînement permet de réaliser de nouveaux services. En effet, cette modélisation se base sur les réseaux de Petri. Ceux-ci présentent suffisamment d'informations pour être exécutables.

De plus, notre modélisation permet non seulement de réaliser un enchaînement de services automatiques (les services du catalogue) mais permet également de rajouter des services manuels qui permettront à l'utilisateur d'interagir lors de l'exécution du *protocole* pour valider, choisir, etc.

Un enchaînement comme un service propose une *description d'IHM*. Cela permet aux utilisateurs débutants d'accéder aisément à ce type de service. Les autres utilisateurs pourront accéder aux services déjà définis mais pourront également créer l'enchaînement qu'ils souhaitent voir réaliser.

Enfin, notre modélisation permet de prendre en compte d'autres caractéristiques comme :

- la *donnée*, qui comprend désormais les commentaires du biologiste, sa provenance, les services réalisés avec cette donnée en entrée, des références vers les bases de données. Toutes ces informations aideront le biologiste à faire évoluer la compréhension qu'il a de sa donnée ;
- les caractéristiques de chaque appel. Cette information pourra être considérée par le biologiste comme étant les conditions d'expérience ;
- les programmes permettant de réaliser plusieurs tâches ;
- les multiples *classifications* qui s'adapteront à la situation de l'utilisateur.

Toutes ces informations permettront de faciliter le travail du biologiste. Il pourra ainsi se concentrer sur ses expériences.

Notre travail n'est qu'une ébauche de modélisation. Il faudrait maintenant mettre en oeuvre cette modélisation afin de vérifier sa réalisation ainsi que ses impacts réels sur l'utilisation des outils bioinformatiques.

D'autres projets ont permis une modélisation du service par la réalisation d'ontologie comme [Wroe *et al.*, 2003] mais ils n'ont pas modélisé les éléments annexes : IHM et protocole. Notre travail par l'intégration de ces différents aspects est donc à notre connaissance unique.

Toutefois, notre modélisation ne constitue qu'un début d'analyse. Dès lors, notre modélisation n'intègre pas la visualisation des données en accord avec notre modélisation. De même, elle ne propose qu'un type de présentation de service.

Bibliographie

[Andersen et Deakin, 2004] Aaron Andersen et Neil Deakin. Xul : Xml user interface language. 2004. http://www.xulplanet.com (Dernière mise à jour : indéterminée) (Dernière visite : 29 août 2004).

[BEN, 2004a] Belgian EMBnet Node : BEN. Bio-computing with ben. 2004. ftp://ben05.ulb.ac.be/pub/BEN-Course/Biocomputing_with_BEN/.

[BEN, 2004b] Belgian EMBnet Node : BEN. Cours introductif à la bioinformatique. 2004. ftp://ben05.ulb.ac.be/pub/BEN-Course/Introductory_course/.

[Bodard, 2002] François Bodard. Ingénierie des interfaces homme/machine. Namur, Belgique, 2002.

[Buyle et Dallons, 2003] Pierre Buyle et Quentin Dallons. Partage de ressources bioinformatiques hétérogènes - conception et implémentation d'une fédération de médiateurs. Pour obtenir une copie contactez http://www.info.fundp.ac.be, Juin 2003.

[Buyle *et al.*, 2004] Pierre Buyle, Quentin Dallons, et Vincent Englebert. Bionformatics grid ressources and environments. 2004. http://www.info.fundp.ac.be/~bdi/Echo_Institut/EchoMai04/EchoMai04.html (Dernière mise à jour : indéterminée) (dernière visite : 29 août 2004).

[Claverie et Notredame, 2003] Jean-Michel Claverie et Cedric Notredame. *Bioinformatics for dummies.* 2003.

[Debaisieux et Desouza, 2003] Laurent Debaisieux et Fernando Desouza. Partage de ressources bioinformatiques hétérogènes. Pour obtenir une copie contactez http://www.info.fundp.ac.be, Janvier 2003.

[EBI, 2004a] European Bioinformatic Institute : EBI. 2can : Bioinformatics educational resource. 2004. http://www.ebi.ac.uk/2can/bioinformatics/index.html (Dernière mise à jour : 11 Mars 2004) (dernière visite : 29 août 2004).

[EBI, 2004b] European Bioinformatic Institute : EBI. Ajax command definition (acd). 2004. http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Acd/(Dernière mise à jour : indéterminée) (Dernière visite : 29 août 2004).

[EBI, 2004c] European Bioinformatic Institute : EBI. European molecular bioinformatic open software suite. 2004. http://www.emboss.org (Dernière mise à jour : indéterminée) (Dernière visite : 29 août 2004).

[EBI *et al.*, 2004] European Bioinformatics Institute : EBI, IT Innovation, the Rosalind Franklin Centre for Genomic Research (RFCGR), Newcastle Computer Science faculty, Newcastle Centre for Life, Manchester Computer Science faculty, et the Nottingham University Mixed Reality Lab. Taverna. 2004. http://taverna.sourceforge.net/ (Dernière mise à jour : indéterminée) (Dernière visite : 29 août 2004).

[Englebert, 2003] Vincent Englebert. Conception des systèmes d'information coopératifs. Namur, Belgique, 2003.

[IBMM/ULB *et al.*, 2004] IBMM/ULB, FUNDP, et IRIDIA/ULB. Bigre. 2004. `http://www.info.fundp.ac.be/~bigre` (Dernière mise à jour : indéterminée) (dernière visite : 29 août 2004).

[LabBook, 2004] LabBook. Bioinformatic sequence markup language. 2004. `http://www.bsml.org` (Dernière mise à jour : indéterminée) (Dernière visite : 29 août 2004).

[UMBER *et al.*, 2004] University of Manchester : UMBER, Swiss Institute of Bioinformatics, University of Nijmegen, University of the Western Cape, European Bioinformatics Institute, Instituto Gulbenkian de Ciencia, University of Bruxelles, Canada Institute for Marine Biosciences, Research Institute for Genetic engineering, Biotechnology, et Expert Center for Taxonomic Identification. European multimedia bioinformatics educational resource **ember**. 2004. `http://www.bioinf.man.ac.uk/ember/` (Dernière mise à jour : indéterminée) (dernière visite : 29 août 2004).

[OMG, 2004] Object Management Group : OMG. Biomolecular sequence analysis (bsa). 2004. `http://www.omg.org/technology/documents/formal/biomolecular_sequence.ht%m` (Dernière mise à jour : 19 août 2004) (Dernière visite : 29 août 2004).

[Pasteur, 2004] Institut Pasteur. Pise : Pasteur institute software environment. 2004. `http://www.pasteur.fr/recherche/unites/sis/Pise/` (Dernière mise à jour : 6 juillet 2004) (Dernière visite : 29 août 2004).

[Sarachu et Colet, 2004] Martin Sarachu et Marc Colet. wemboss. 2004. `http://www.wemboss.org/` (Dernière mise à jour : 25 février 2004) (Dernière visite : 29 août 2004).

[van der Aalst et van Hee, 2002] Wil van der Aalst et Kees van Hee. *Workfow management.* 2002.

[Wroe *et al.*, 2003] Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, et Mark Greenwood. a suite of daml+oil ontologies to describe bioinformatics web services and data. *International Journal of Cooperative Information Systems*, 12:197–224, 2003.

Troisième partie

Annexes

Annexe A

Techniques d'alignement et de recherche par similarité

Pour comprendre l'intérêt d'utiliser tel programme plutôt que tel autre, il est important de comprendre les différences existant entre les différents programmes. Les explications proviennent de : [BEN, 2004b].

Ce chapitre ne porte que sur l'étude des techniques d'alignement et de recherche de similarité car elles sont utilisées dans le cadre de notre fil rouge. Nous n'allons développer que les techniques d'alignement provenant du package EMBOSS[EBI, 2004c] et le programme BLAST réalisant la recherche de similarité.

A.1 L'alignement

Un alignement est un processus où l'on cherche à trouver les événements qui ont pu transformer une séquence source en séquence cible. Les transformations sont soit des substitutions, soit des insertions, soit des suppressions ou "deletions". Voici un exemple d'alignement (voir figure A.1).

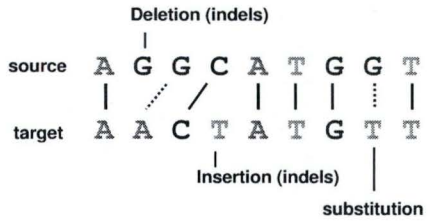


FIG. A.1 – *Principe de l'alignement [BEN, 2004b]*

La représentation informatique de cet alignement est présentée à la figure A.2. Les insertions et deletions sont représentées par des caractères - appelés "gap". Les caractères entre les séquences permettent de voir si il y a eu mutation : ".", si la correspondance est parfaite : "|".

```
########################################
# Program:  needle
# Rundate:  Fri Apr 09 23:59:25 2004
# Align_format: srspair
# Report_file: source.needle
########################################
#=======================================
#
# Aligned_sequences: 2
# 1: source
# 2: target
# Matrix: ENUC.4.2
# Gap_penalty: 5.0
# Extend_penalty: 4.0
#
# Length: 10
# Identity:        6/10 (60.0%)
# Similarity:      6/10 (60.0%)
# Gaps:            2/10 (20.0%)
# Score: 12.0
#
#
#=======================================

source             1 AGGC-ATGGT      9
                       | .| |||.|
target             1 A-ACTATGTT      9


#---------------------------------------
#---------------------------------------
```

FIG. A.2 – *Résultat d'un programme d'alignement*
résultat du programme *needle* (package EMBOSS) (réalisé auprès du fournisseur : `http://www.be.embnet.org/wEMBOSS/`)

En effet, assez rapidement les bioinformaticiens ont réalisé des programmes permettant de trouver une mutation entre deux séquences. Ces programmes génèrent une série de mutations, évaluent leur probabilité de façon approximative au moyen d'un score et affichent la mutation qui a probablement eu lieu ainsi que le score réalisé (dans l'exemple le score est de 12).

Malheureusement, il n'existe pas qu'un seul algorithme qui permette de toujours trouver cette mutation. Les bioinformaticiens ont développé une série d'outils, chacun adapté à une situation précise. Nous avons fait différentes distinctions sur ces outils.

La première distinction se base sur la méthode de l'outil, autrement dit sur le produit de l'outil. La seconde distinction se base sur la technique de l'outil ou algorithme, autrement dit la base de son calcul. La dernière distinction se base sur le programme proprement dit.

A.1.1 Les différentes méthodes

Pour comparer deux séquences, il existe différentes méthodes. Ces méthodes produisent différents types de résultats : un graphique (dot plot) ou encore un alignement (comme la figure A.2).

Utiliser des méthodes différentes est intéressant. En effet, les différents types de résultats apportent des informations complémentaires.

Pour l'alignement, les méthodes disponibles sont :

- le dot plot (figure A.3). C'est un graphique représentant les zones de similarité, chaque segment correspond à un alignement. Il permet d'avoir une vue d'ensemble.

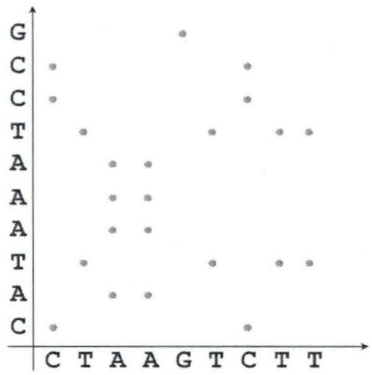


FIG. A.3 – *Dot plot [BEN, 2004b]*

Le graphe présente la fonction :

$$f(x,y) = 1 \; si \; x = y, \; 0 \; sinon$$

où la séquence 1 est placée en abscisse et la séquence 2 en ordonnée.

- l'alignement global (figure A.4). Cette méthode aligne les deux séquences sur tout leur longueur.

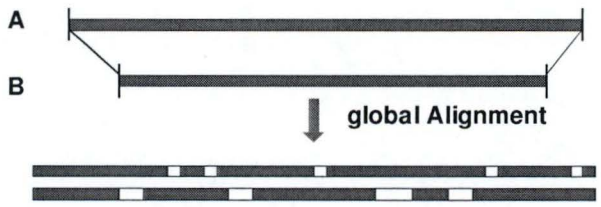


FIG. A.4 – *Alignement global [BEN, 2004b]*

Pour réaliser l'alignement, le programme calcule le score en se basant sur :

- une matrice de substitution s : matrice qui indique le coût du changement d'un acide aminé par un autre, en effet à cause de certaines propriétés chimiques, certaines mutations sont plus observées que d'autres (exemple : figure A.6).

- deux entiers représentant le coût de l'ouverture et de l'extension d'un gap. Le gap est le trou qui met en évidence l'insertion ou la délétion. Il est important de distinguer l'ouverture de l'extension, car dès qu'une rupture se crée dans la chaîne (ouverture), il est plus aisé pour d'autres nucléotides ou acides aminés de s'insérer (extension).
  Pour permettre la compréhension de l'algorithme, nous avons simplifié l'algorithme : un seul paramètre d correspond au coût d'un gap (le coût d'un gap correspond à la pénalité introduite par une cassure dans la chaîne).

Le calcul du score se base sur une matrice : chaque élément de la matrice correspond au score de l'alignement pour le segment de 1 à i éléments de la séquence source et de 1 à j pour la séquence cible.

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x^i,y^j) & \textit{élément identique ou mutation} \\ F(i-1,j) - d & \textit{insertion d'un gap dans la séquence source} \\ F(i,j-1) - d & \textit{insertion d'un gap dans la séquence cible} \end{cases}$$

Les bords de cette matrice sont calculés par : $F(i,0) = -id$, $F(0,j) = -jd$

Le choix de l'élément offrant le maximum va permettre de réaliser un chemin dans cette matrice (d'après l'élément sur lequel se base le calcul).
Le score final est le dernier élément : en bas à droite. A partir de cet élément, nous allons retrouver les choix (mutation, insertion, délétion) qui vont ainsi permettre de réaliser l'alignement des deux séquences (voir exemple en fin de section).

- l'alignement local (figure A.5). Cette méthode aligne les deux séquences sur leur(s) partie(s) la(les) plus similaire(s) et ignore les parties peu similaire.

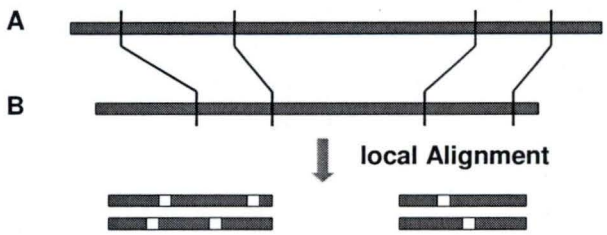


FIG. A.5 – *Alignement local [BEN, 2004b]*

Le principe est semblable à celui de l'alignement global. Seule la fonction est différente pour permettre de prendre en compte un alignement d'un segment des séquences.

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x^i,y^j) & \textit{élément identique ou mutation} \\ F(i-1,j) - d & \textit{insertion d'un gap dans la séquence source} \\ F(i,j-1) - d & \textit{insertion d'un gap dans la séquence cible} \\ 0 & \textit{début de l'alignement} \end{cases}$$

Les bords de cette matrice sont calculés comme $F(i,0) = 0$, $F(0,j) = 0$.

Chacune de ces méthodes va donc permettre à l'utilisateur de visualiser la même information (similitude entre ces deux séquences) mais de manière très différente.

On conseille, en général, à l'utilisateur de faire d'abord un dot plot pour se rendre compte de son degré de similitude et un alignement local tant qu'il ne sait pas si ses séquences s'alignent sur toute leur longueur.

Exemple : réalisation d'un alignement global entre HEAGAWGHEE et PAWHEAE.

La figure A.6 représente la matrice de substitution utilisée pour ce calcul. La coût d'un gap est de 10.

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

FIG. A.6 – *Exemple de matrice de substitution [BEN, 2004b]*

La figure A.7 représente la matrice qui permet d'obtenir l'alignement optimal. Deux chemins équivalents permettent de réaliser le score maximum : -8. Ces deux alignements sont présentés dans les deux cadres rouges.

		H	E	A	G	A	W	G	H	E	E
	0	-10	-20	-30	-40	-50	-60	-70	-80	-90	-100
P	-10	-2	-11	-21	-31	-41	-51	-61	-71	-81	-91
A	-20	-12	-3	-6	-16	-26	-36	-46	-56	-66	-76
W	-30	-22	-13	-6	-9	-19	-11	-21	-31	-41	-51
H	-40	-20	-22	-15	-8	-11	-21	-13	-11	-21	-31
E	-50	-30	-14	-23	-18	-9	-14	-23	-13	-5	-14
A							-12	-14	-23	-14	-6
E	-70	-50	-34	-19	-11	-20	-16	-15	-14	-17	-8

HEAGAWGHEE
---PAWHEAE

HEAGAWGHEE
-P--AWHEAE

FIG. A.7 – *Exemple de matrice permettant de calculer un alignement global [BEN, 2004b]*

A.1.2 Les différentes techniques

Pour chacune de ces méthodes, différentes techniques existent. Ces techniques ont été développées pour adapter la méthode à la taille des données, aux types de données en entrée,...

Ainsi, pour l'alignement local, il existe :

- une technique précise mais gourmande en temps et en CPU (algorithme de Smith-Waterman : basé sur la matrice présenté ci-dessus).
- une version se basant sur des heuristiques (blast voir ci-après). Elle est un peu moins précise mais plus rapide.

De même que pour le dot plot, deux techniques ont été développées : la technique "word" et la technique "window stringency" (voir figure A.8).

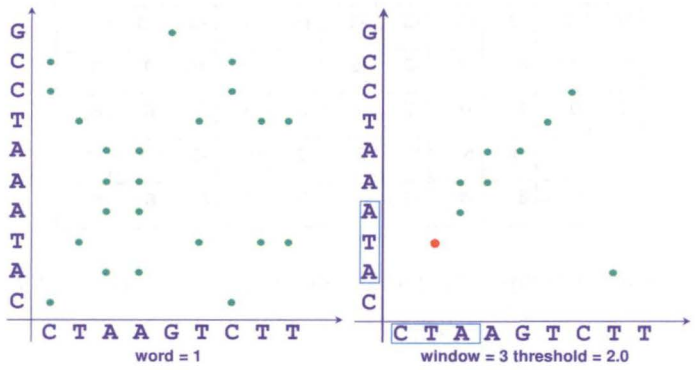


FIG. A.8 – *Différentes techniques de dot plot [BEN, 2004b]*

La première méthode place une croix si les mots sont identiques. La séquence est découpée en mots. Une lettre n'appartient qu'à un mot. Un paramètre essentiel de cette méthode est la taille du mot.

La seconde méthode vérifie que les morceaux de séquences sont similaires (pas exactement identiques). Celle-ci se base sur la "sliding window" (voir figure A.9). On aligne chaque fenêtre de la séquence 1 contre toutes les fenêtres de la séquence 2 et si le score dépasse la stringency, un point est mis sur le graphique.

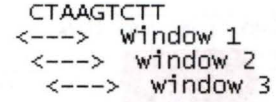


FIG. A.9 – *Technique de "splicing window" [Claverie et Notredame, 2003]*

Ces différentes techniques ont un impact sur le résultat. L'utilisateur doit donc savoir s'il préfère un alignement précis ou s'il accorde plus d'importance au temps de réponses.

Dans le cas du dot plot, l'utilisateur doit utiliser la technique window/stringency lorsque les séquences sont éloignées.

A.1.3 Les différents programmes

Dans la suite EMBOSS [EBI, 2004c], on trouve trois programmes permettant de réaliser un alignement global (algorithme Needleman-Wunsch) : est2genome, needle et stretcher.

Il est préférable d'utiliser :

- needle, lorsque la séquence est courte (moins de 10Kb) car sa complexité est de O(n).
- stretcher lorsque la séquence est grande parce que sa complexité est moindre.
- est2genome, ce programme a été spécifiquement conçu pour comparer des EST [1] à un génome [2].

La différence entre needle et stretcher est similaire à la différence entre water et matcher pour l'alignement local réalisé au moyen de l'algorithme Smith-Waterman.

Ces différences accroissent le nombre d'outils pour réaliser un alignement, ce qui fait que beaucoup de biologistes n'utilisent pas le meilleur programme.

A.2 Recherche de similarité auprès de bases de données

Ces programmes permettent à l'utilisateur d'obtenir une liste de séquences similaires à sa séquence requête. Pour ce faire, ces programmes réalisent un alignement pairwise de toutes les séquences de la base de données contre la séquence requête, il ne présente à l'utilisateur que les correspondances les plus significatives.

Forcément, ces outils sont en forte connexion avec les techniques d'alignement (principalement local). On retrouve des outils se basant sur les techniques : Smith-Waterman et blast.

Nous allons nous intéresser seulement à l'outil BLAST (Basic Local Alignment Search Tools). Celui-ci est un des programmes les plus rapides réalisant une recherche par similarité.

Cet algorithme peut être découpé en étapes :

1. Création d'une liste de mots de longueur w avec les plus grand score. ($w = 3$, pour les protéines, 11 pour les nucléotides)(voir A.10).

1. EST signifie Expressed Sequence Tags. C'est une séquence partielle d'ADN de quelques centaines de nucléotides provenant d'un ARN. On l'utilise plutot que l'ARN car il offre l'immense avantage d'être plus stable que la molécule d'ARN et de pouvoir être stocké, copié et séquencé.

2. Un génome est une séquence d'ADN codant pour une protéine

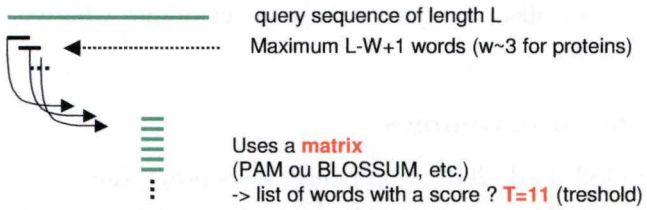


FIG. A.10 – *BLAST : première étape [BEN, 2004b]*

La séquence requête est divisée en mots. Le programme ne retient que les mots les plus significatifs, rares (ce calcul se base sur une matrice de substitution - correspond à un score).

2. Comparaison : liste de mots / base de donnée → exact match (voir A.11).

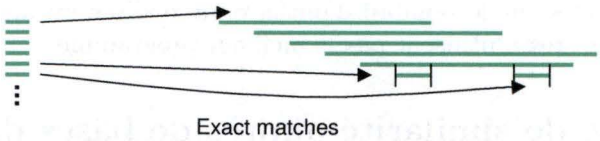


FIG. A.11 – *BLAST : seconde étape [BEN, 2004b]*

On compare maintenant ces mots avec la base de données. Seules les correspondances exactes sont retenues.

3. Extension de ces correspondances et réalisation du meilleur alignement local (voir A.12).

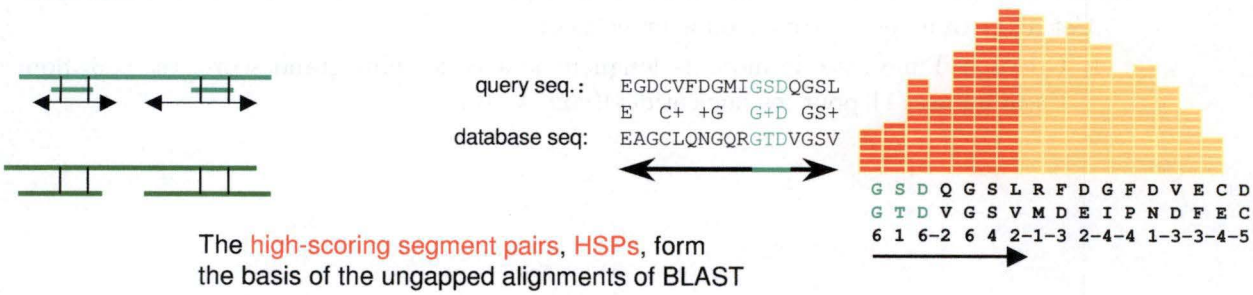


FIG. A.12 – *BLAST : dernière étape [BEN, 2004b]*

A partir de ces séquences provenant de la base de données, le programme essaye d'étendre cette zone en alignant le reste de la séquence. Il ne garde que les séquences ayant un score dépassant un certain seuil.

4. Calcul pour chaque séquence similaire de l'E-value.

L'E-value correspond au nombre d'alignements différents avec un score équivalent ou plus élevé qui serait considéré par BLAST comme similaire par hasard. Plus sa valeur est petite plus le score est significatif et donc la probabilité que ces séquences soient similaires est élevée.

Ces différentes étapes permettent d'éviter le nombre de comparaisons trop important. Ce programme peut également être utilisé pour comparer deux séquences.

Annexe B

Application de notre modélisation à l'exemple fil rouge

Dans notre chapitre sur la bioinformatique, nous avons introduit une expérience basique (voir section 1.3 page 12). Cet exemple sera modélisé au moyen des différents outils définis dans ce mémoire. Il sera ainsi défini comme un nouveau service.

Cette modélisation a été réalisée en trois étapes, nous avons d'abord défini rapidement le protocole (ou l'enchaînement des services), pour ensuite nous intéresser au service (non seulement le service final, mais les différents services utilisés dans le protocole). Enfin, nous aborderons l'apparence de ce service par la modélisation de l'IHM.

B.1 Le protocole

Dans l'exemple d'utilisation présenté, l'utilisateur réalisait d'abord une recherche de séquences similaires. Ensuite, il sélectionnait les données qui lui semblaient intéressantes. Il devait ensuite se rendre auprès d'un autre fournisseur. Là, il pouvait réaliser un alignement et un dot plot qui lui permettait de comparer les données intéressantes avec la donnée en entrée.

On peut définir cet enchaînement comme un service au moyen de notre modélisation. La figure B.1 représente ce service : le service calcule d'abord les séquences similaires à une donnée en entrée. Ensuite, il demande à l'utilisateur de sélectionner les données intéressantes. Enfin, il compare les données sélectionnées à la séquence en entrée au moyen d'un alignement local et d'un dot plot.

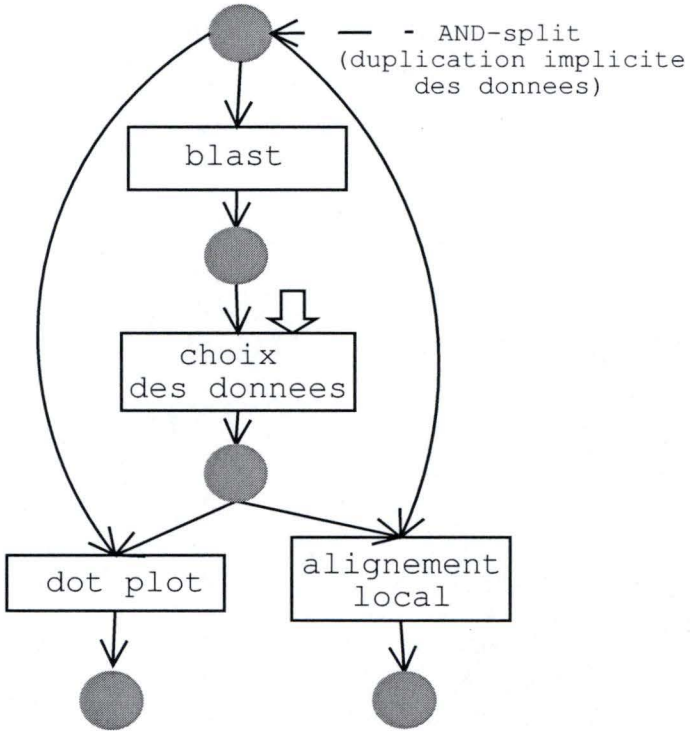


FIG. B.1 – *Représentation dynamique du protocole* blast et approfondissement de certains résultats

B.2 Les services

Dans cette section, nous allons d'abord étudier les différents services utilisés dans le protocole. Ceux-ci sont soit automatiques, soit manuels. Enfin, nous aborderons la signature du service que nous venons de créer.

B.2.1 Les services automatiques utilisés

Dans le protocole (voir figure B.1), trois services automatiques sont présentés : `blast`, `alignement local` et `dot plot`.

`Blast` est modélisé par la figure B.2 (également par la figure 5.9 page 60). C'est un exemple de service concret se basant sur une implémentation capable d'offrir plusieurs services.

Service (concret)
réalisé par l'implémentation BLAST
- id : S12375
- nom : BLAST
- description : service permettant de trouver des séquences similaires dans une base de données
- traitement : recherche par similarité
- algorithme : BLAST
- précision : faible
- signature "niveau débutant" :
  f(c {blastp, blastn}, x sequence, y sequenceBD [format = BLAST format]) = z listesequence
- signature "niveau normal" :
  f(c {blastp, blastn}, x sequence, y sequenceBD [format = BLAST format], expect_value int, word_size int) = z listesequence

FIG. B.2 – *Modélisation du service BLAST*

`Alignement local` est modélisé par la figure B.3 (également par la figure 5.15 page 66). C'est un exemple de service abstrait qui est réalisé par deux services concrets : water et matcher (deux programmes du package EMBOSS). Lors de l'exécution de ce service, le système dispatchera la demande d'exécution vers un de ces deux services d'après la taille de la donnée[1].

Service (abstrait)
est réalisé par les services concrets : water et matcher
- id : S12401
- nom : smith-Waterman
- description : alignement de deux séquences au moyen de l'algorithme de Smith-Waterman
- traitement : alignement pairwise local
- algorithme : Smith-Waterman
- précision : élevée
- signature "niveau normal" :
  f(sequence 1 sequence, sequence 2 sequence, gap opening penalty int, gap extension penalty int, matrice de substitution matrix) = résultat alignement(s)

FIG. B.3 – *Modélisation du service algorithme Smith-Waterman (alignement local)*

`Dot plot` est modélisé par la figure B.4. Ce service correspond en fait au programme dotmatcher du package EMBOSS.

1. Le service alignement local sera réalisé au moyen de water pour les séquences de petite taille ou au moyen de matcher pour les séquences de grande taille.

Service (concret)
réalisé par l'implémentation dotmatcher
- id : S12526
- nom : dotmatcher
- description : aligne les séquences au moyen d'un graphique (dot plot).
- traitement : alignement pairwise dot plot
- algorithme : window/stringency
- précision : élevée
- signature "niveau normal" :
  f(sequence1 sequence, sequence2 sequence, window_size int, threshold real) = résultat graphique

FIG. B.4 – *Modélisation du service dotmatcher (dot plot)*

B.2.2 Les services manuels utilisés

Le service manuel permet à l'utilisateur de sélectionner les données qui lui semblent intéressantes. La relation entre ces données et la séquence en entrée sera approfondie au moyen d'un alignement plus précis et un dot plot.

On peut définir ce service comme :

$$choix\ des\ données : listeSequence \longrightarrow listeSequence$$

Où le résultat correspond à un filtrage effectué par l'utilisateur de la liste de séquences données en entrée.

B.2.3 Le service global

Le protocole est également défini comme un service. Sa description lui est propre mais la signature se base sur la signature de ses composants. La figure B.5 représente la modélisation de ce service.

Service (abstrait)
est réalisé par les services concrets : blast, (water et matcher), dotmatcher

- id : S24851
- nom : recherche de séquence similaire
- description : réalisation d'une recherche par similarité accompagnée d'un approfondissement de la similarité de certaines séquences.
- traitement : recherche par similarité
- précision : moyenne
- signature "niveau normal"
  f(x sequence, y sequenceBD [format = BLAST format], donnée-choisie liste_sequence, c {blastp, blastn}, expect_value int, word_size int, gap opening penalty int, gap extension penalty int, matrice de substitution matrix, window_size int, threshold real) = z listesequence, résultat-alignement alignement(s), résultat-dot-plot graphique

FIG. B.5 – *Modélisation du service* blast et approfondissement de certains résultats

B.3 Les Interfaces Homme Machine

Les services ont été définis, il reste maintenant à décrire l'interface de ces services. Nous allons d'abord décrire les visualisateurs des différents services pour ensuite étudier la conversation.

B.3.1 Les visualisateurs

La figure B.6 est un visualisateur du service BLAST.

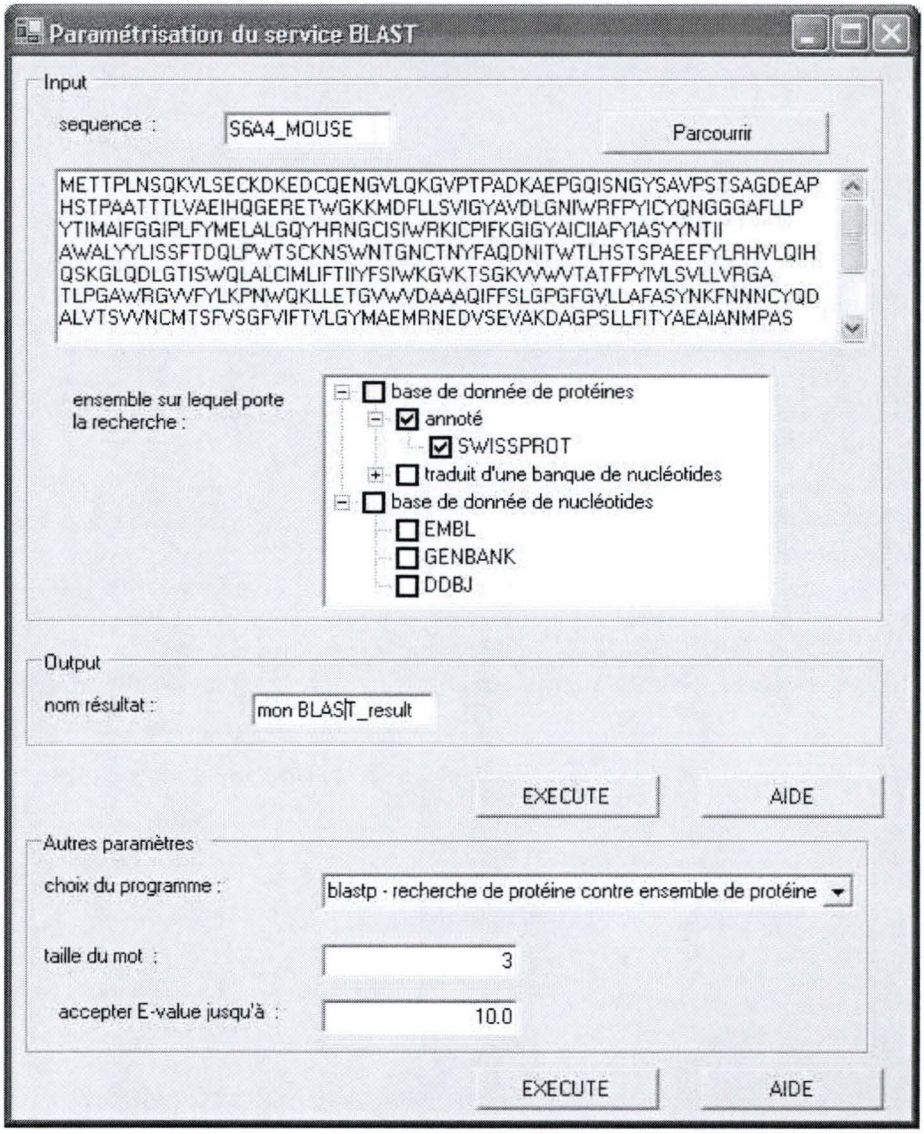
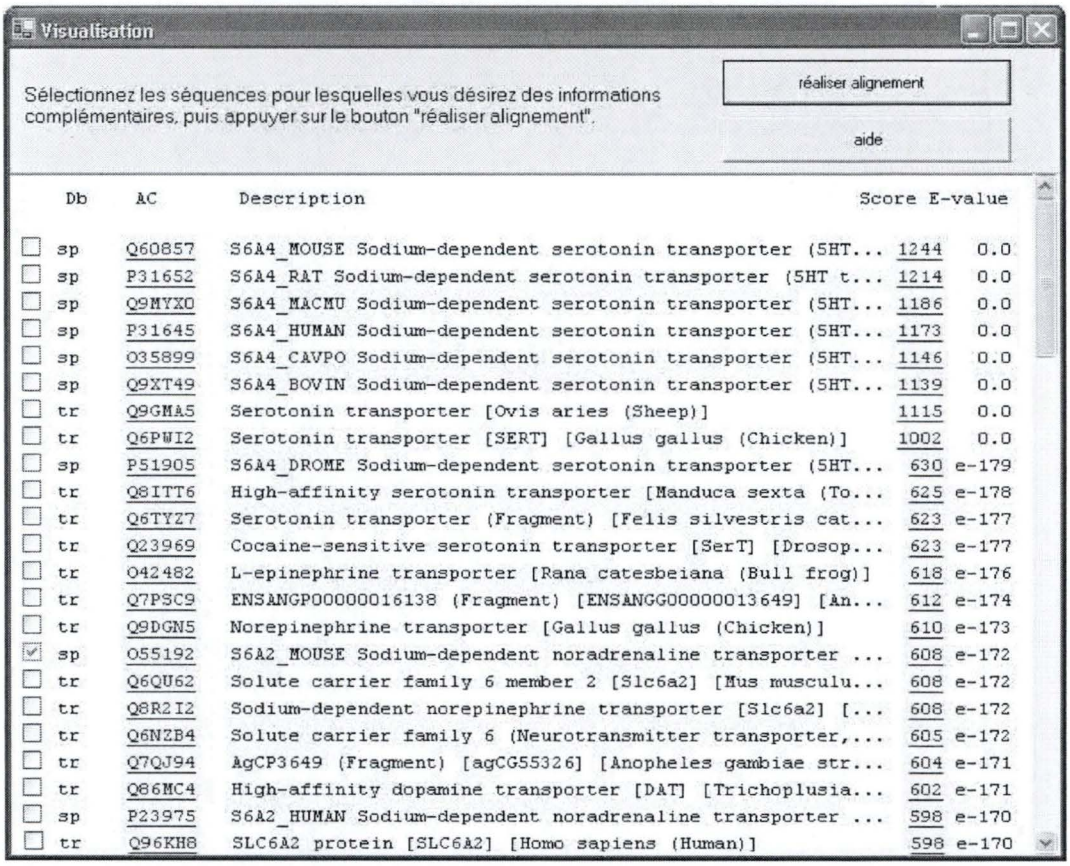


FIG. B.6 – *Présentation du service BLAST*

La figure B.7 permet à l'utilisateur de sélectionner les données provenant du résultat de l'exécution du service BLAST.



	Db	AC	Description	Score	E-value
☐	sp	Q60857	S6A4_MOUSE Sodium-dependent serotonin transporter (5HT...	1244	0.0
☐	sp	P31652	S6A4_RAT Sodium-dependent serotonin transporter (5HT t...	1214	0.0
☐	sp	Q9MYX0	S6A4_MACMU Sodium-dependent serotonin transporter (5HT...	1186	0.0
☐	sp	P31645	S6A4_HUMAN Sodium-dependent serotonin transporter (5HT...	1173	0.0
☐	sp	O35899	S6A4_CAVPO Sodium-dependent serotonin transporter (5HT...	1146	0.0
☐	sp	Q9XT49	S6A4_BOVIN Sodium-dependent serotonin transporter (5HT...	1139	0.0
☐	tr	Q9GMA5	Serotonin transporter [Ovis aries (Sheep)]	1115	0.0
☐	tr	Q6PWI2	Serotonin transporter [SERT] [Gallus gallus (Chicken)]	1002	0.0
☐	sp	P51905	S6A4_DROME Sodium-dependent serotonin transporter (5HT...	630	e-179
☐	tr	Q8ITT6	High-affinity serotonin transporter [Manduca sexta (To...	625	e-178
☐	tr	Q6TYZ7	Serotonin transporter (Fragment) [Felis silvestris cat...	623	e-177
☐	tr	Q23969	Cocaine-sensitive serotonin transporter [SerT] [Drosop...	623	e-177
☐	tr	O42482	L-epinephrine transporter [Rana catesbeiana (Bull frog)]	618	e-176
☐	tr	Q7PSC9	ENSANGP00000016138 (Fragment) [ENSANGG00000013649] [An...	612	e-174
☐	tr	Q9DGN5	Norepinephrine transporter [Gallus gallus (Chicken)]	610	e-173
☑	sp	O55192	S6A2_MOUSE Sodium-dependent noradrenaline transporter ...	608	e-172
☐	tr	Q6QU62	Solute carrier family 6 member 2 [Slc6a2] [Mus musculu...	608	e-172
☐	tr	Q8R2I2	Sodium-dependent norepinephrine transporter [Slc6a2] [...	608	e-172
☐	tr	Q6NZB4	Solute carrier family 6 (Neurotransmitter transporter,...	605	e-172
☐	tr	Q7QJ94	AgCP3649 (Fragment) [agCG55326] [Anopheles gambiae str...	604	e-171
☐	tr	Q86MC4	High-affinity dopamine transporter [DAT] [Trichoplusia...	602	e-171
☐	sp	P23975	S6A2_HUMAN Sodium-dependent noradrenaline transporter ...	598	e-170
☐	tr	Q96KH8	SLC6A2 protein [SLC6A2] [Homo sapiens (Human)]	598	e-170

FIG. B.7 – *Présentation du service* choix des données

La figure B.8 est un visualisateur composite qui permet à l'utilisateur d'interagir avec le service alignement local et le service dot plot. Proposer ce type d'interface est intéressant lorsque ces deux services sont réalisés à partir des mêmes données en entrée (dans notre exemple : deux séquences de même type).

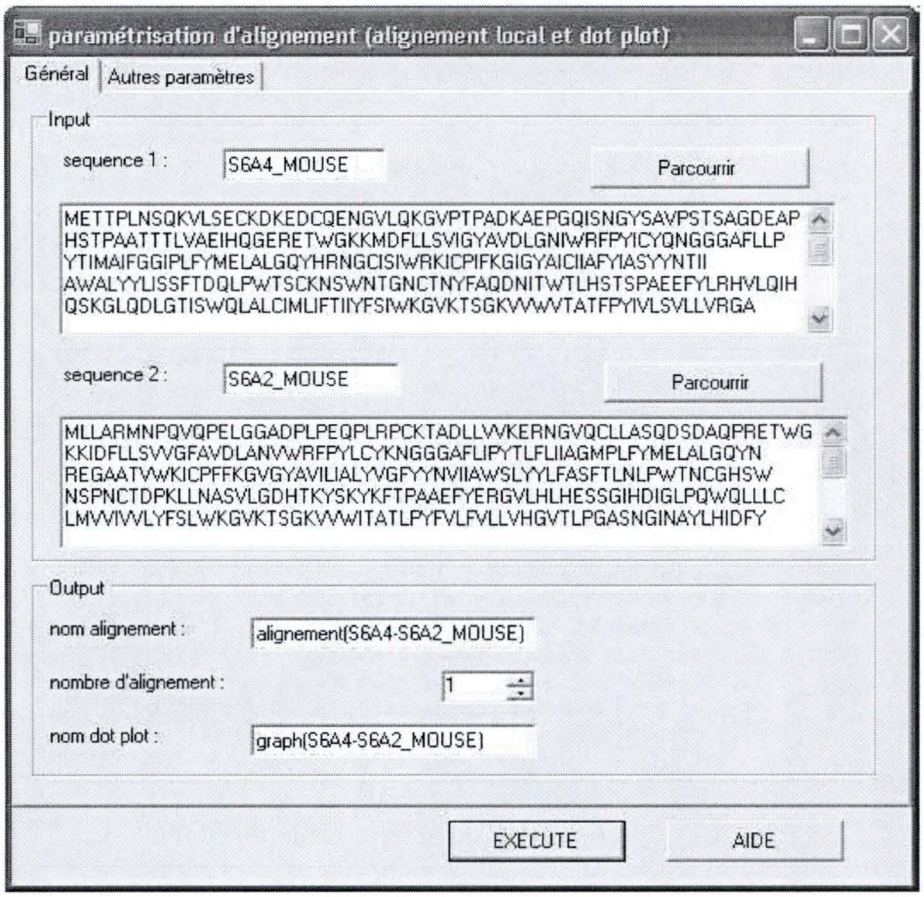


FIG. B.8 – *Présentation des services alignement local et dot-plot (premier onglet)*

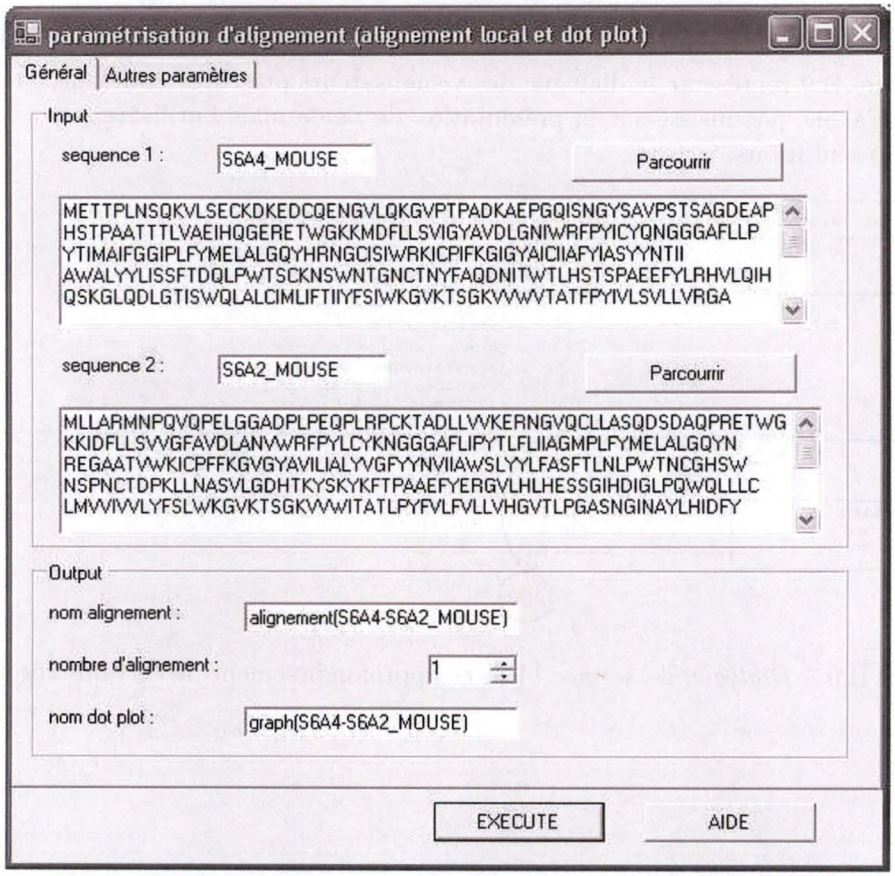


Figure B.8 Présentation des services alignement local et dot-plot (deuxième onglet)

B.3.2 Dialogue

La figure B.9 représente le dialogue des visualisateurs présentés ci-dessus.

Nous n'avons pas insisté sur la présentation de l'aide mais l'utilisateur y a accès dans les trois présentations.

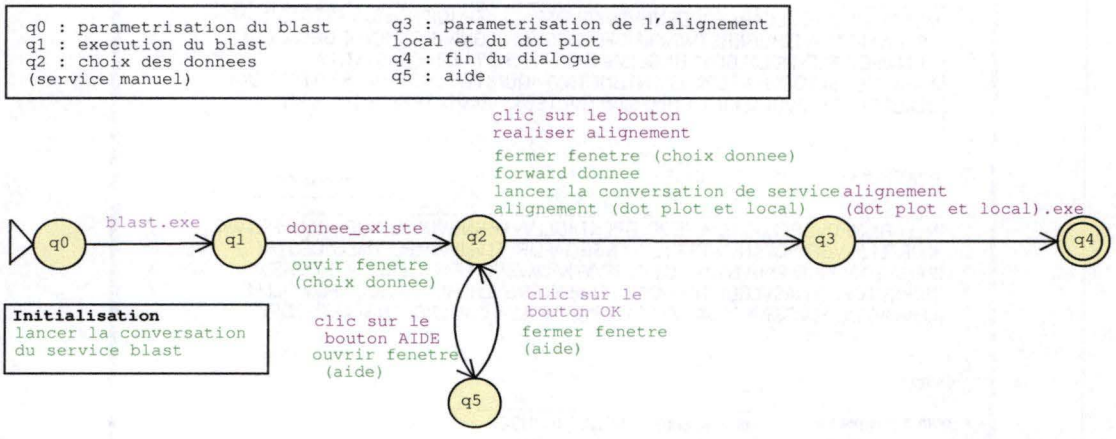


FIG. B.9 – *Dialogue du service* blast et approfondissement de certains résultats